SIGGRAPH 2001 Course Notes

Digital Geometry Processing

Organizers: Wim Sweldens, Bell Labs
Peter Schröder, Caltech

Summary

The arrival of 3D scanning has created a new wave of digital media after sound, images, and video, rasing the need for digital processing algorithms. Traditionally fine detail geometry is represented through unstructured polygonal meshes. Such meshes are ackward for editing, filtering, and compression applications. In this course we propose a new paradigm based on semi-regular meshes, constructed through a process of recursive quadrisection. Several research results have shown their many advantages. We will show how to build semi-regular meshes from unstructured polygonal meshes and raw range data, and how to build applications such as filtering, editing, simulation, and compression using semi-regular meshes.

Lecturers

Brian Curless
Department of Computer Science & Engineering
University of Washington
Seattle, Washington
curless@cs.washington.edu

Igor Guskov
Computer Science Department
California Institute of Technology
Pasadena, CA
ivguskov@cs.caltech.edu

Peter Schröder
Computer Science Department
California Institute of Technology
Pasadena, CA
ps@cs.caltech.edu

Wim Sweldens
Bell Laboratories
Lucent Technologies
Murray Hill, NJ
wim@lucent.com

Denis Zorin
Media Research Laboratory
New York University
New York, NY
dzorin@mrl.nyu.edu

Morning Schedule

8h30-9h00: Introduction
Wim Sweldens

9h00-10h00: Acquisition
Brian Curless

10h00-10h15: Break

10h15-11h00: Remeshing I: Basics
Wim Sweldens

11h00-12h00: Remeshing I: Details
Igor Guskov

Afternoon Schedule

1h30-2h00: Subdivision
Peter Schrder

9h00-10h00: Processing
Denis Zorin

3h00-3h15: Break

3h15-4h00: Compression
Peter Schrder

4h00-5h00: Demos All

Lecturers' Biographies

Brian Curless is an assistant professor of Computer Science and Engineering at the University of Washington. Curless's research has focused on acquiring and building complex geometric and appearance models using structured light scanning systems. While earning his PhD at Stanford he developed fundamentally better methods for optical triangulation and a new method for combining range images that led to the first "3D fax" of a geometrically complex object. He later worked with Marc Levoy on the Digital Michelangelo Project in Florence where they captured the geometry and appearance of a number of Michelangelo's statues. Recently, his work has emphasized appearance capture as described in SIGGRAPH papers on "environment matting" and "surface light fields." Curless currently sits on the Technical Advisory Board for Paraform, Inc., a company that is commercializing Stanford-developed technology for building CAD-ready models from range data and polygonal meshes. He has taught at the undergraduate and graduate levels, including courses related to 3D photography co-organized at Stanford, the University of Washington, CVPR '99, SIGGRAPH '99, and SIGGRAPH '00. Curless received a university-wide Outstanding Teaching Award from Stanford University and an NSF CAREER award and Sloan Fellowship at the University of Washington.

Igor Guskov is currently a Research Rellow at the Caltech Department of Computer Science. His research interests lie in the general area of multiresolution methods for efficient processing of meshes representing complex geometric shapes. He received his BS degree from Moscow State University, and his PhD in applied mathematics from Princeton University. In his thesis Igor introduced "irregular subdivision," a crucial component for building wavelet representations on irregular mesh hierarchies. This wavelet transform is very useful for denoising, enhancement, filtering, and editing of irregular meshes. More recently, he co-invented Normal Meshes and a remeshing procedure producing meshes whose geometry is represented with just one float per vertex. These meshes are ideally suited for further compression. Igor is also interested in interactive geometry creation and applications of wavelets in scientific computing. Among many other venues he has published several papers in Siggraph.

Peter Schröder is an associate professor of computer science at the California Institute of Technology where he directs the Multi-Res Modeling Group. He received his PhD from Princeton University in computer science in 1994. For the past 8 years his work has concentrated on exploiting wavelets and multiresolution techniques to build efficient representations and algorithms for many fundamental computer graphics problems. Most notably he has been involved in the development of many fundamental

algorithms in Digital Geometry Processing. He has published extensively in the Siggraph conferences and taught in a number of Siggraph courses. Most recently he co-led the very successful courses on "Wavelets in Computer Graphics" (1996) and "Subdivision for Modeling and Animation" (1998-2000). His current research focuses on subdivision as a fundamental paradigm for geometric modeling and rapid manipulation of large, complex geometric models. Among many honors he has received for his work are an NSF CAREER award, Okawa Foundation award, a Sloan Fellowship, and a Packard Foundation Fellowship. He is a frequent international lecturer and consultant to industry.

Wim Sweldens received his PhD in 1994 from the Katholieke Universiteit Leuven, Belgium. He is currently the director of Scientific Computing Research at Bell Laboratories, Lucent Technologies. His research is concerned with wavelets and multiscale analysis and its application in numerical analysis, signal processing, computer graphics, and wireless communications. He is the inventor of the lifting scheme, a new design and implementation technique for wavelets on which the JPEG2000 standard is based. Lifting also allows the generalization of multiresolution signal processing techniques to complex geometries. More recently he worked on parameterization, remeshing, and compression of polygonal surfaces. He has lectured internationally on the use of multiscale techniques in computer graphics and receiver several best paper awards; he co-organized the Wavelet course in 96 and participated in three other SIGGRAPH courses. MIT's Technology Review recently selected him as one of 100 top young technological innovators. He is the founder and Editor-in-Chief of the Wavelet Digest.

Denis Zorin is an assistant professor at the Courant Institute of Mathematical Sciences, New York University. He received a BS degree from the Moscow Institute of Physics and Technology, a MS degree in Mathematics from Ohio State University and a PhD in Computer Science from the California Institute of Technology. In 1997-98, he was a research associate at the Computer Science Department of Stanford University. His research interests include multiresolution modeling, the theory of subdivision, and applications of subdivision and multiresolution surfaces in computer graphics, computer-aided design, and scientific computing. He is also interested in perceptually-based computer graphics algorithms and non-photorealistic rendering. In 2000, he was named a Sloan Foundation Fellow. He has published a number of papers in Siggraph proceedings; in 1998-2000 he co-organized the highly successful SIGGRAPH course "Subdivision for Modeling and Animation" attended by 400-500 attendees each year.

Digital Geometry Processing

Peter Schröder Wim Sweldens

Introduction

Multi-media has seen three waves so far: sound, images, and video. We are presently witnessing the arrival of the fourth wave of digital multi-media: geometry.

Each one of these waves was initiated by increases in acquisition capabilities, compute power, storage capacity, and transmission bandwidth for the respective type of data. As Moore's law moved along, we first saw digital sound in the 70's, digital images in the 80's, and digital video in the 90's. Soon an average PC will be able to handle digital geometry.

Each new digitization wave brings with it the need for new processing tools. Typical elements of a signal processing toolbox are: denoising, compression, transmission, enhancement, detection, analysis, editing, etc. While analog circuitry can only handle the most basic processing tasks, as soon as the data is digitized a whole new realm of algorithms becomes feasible. This has lead to an explosion in digital signal processing research, aimed at the development of suitable mathematical representations, their manipulation and associated computational paradigms.

One example is the ubiquitous use of digital sound from portable CD players to musical instruments and digital cellular phones. More recently, increasing network bandwidth and compute power have contributed to a revolution in the distribution of music over networks and on personal computers. Similar observations can be made about images and video.

In a sense, the cheap and plentiful availability of a data type has led to a spur in the development of methods to wring usefulness from this data, which in turn has stimulated progress in the underlying technology to support the respective type of data.

In much a similar way we are now witnessing the arrival of a new data type, 3D geometry. To be sure, geometric modeling has been around for a long time, but geometry was created "by hand" in a tedious custom process. Strides in the acquisition of 3D geometry through low and high end 3D scanners is now making digital proxies of geometry available for processing in a much broader way. Example sources of such geometry range from the sculptures of Michelangelo[1], to manufactured products[2] and the earth itself.[3]

Once again we need to build a toolbox of fundamental algorithms and mathematics to process this new class of digital geometry data. This time the task is *significantly* more challenging than before.

Fourier or not Fourier

Multi-media data of the first three waves can be modeled easily as being defined on a section of Euclidean geometry. Sound is defined as a function of time, i.e., on a 1D line. Similarly, images are naturally defined as functions over a section of a 2D plane. Finally, video is most naturally modeled as a function over a section of 3D, two dimensions in space and one in time. Another useful abstraction is that of sampling. A given datatype is digitized by sampling it at regular intervals (either space or time) and converting the measured values into binary numbers. The act of sampling and the regular spacing between samples make Fourier analysis and the Fourier transform the quintessential tools to understand and manipulate the content of such signals. If the underlying mathematical space where not Euclidean the regular spacing of samples could not be achieved. For example, consider a sphere. There are no equi-spaced sampling patterns on the sphere beyond those given by the five Platonic solids. A digital image produced by one of today's cameras on the other hand is a regular matrix of picture elements, each representing a sample of the image irradiance on the 2D image plane.

[1] Digital Michelangelo Project, Stanford University; models with $> 10^9$ samples.

[2] Reverse engineering and non-invasive inspection.

[3] See the recent shuttle radar topography mission, for example.

This regular sampling allows the computation of the Fast Fourier Transform (FFT), one of the most popular algorithms in digital signal processing. Once in the Fourier domain it is easy to remove noise, for example, or to enhance the image[4]. Recently new multiresolution and time-frequency based methods such as wavelets have proven to be a valuable alternative to the Fourier transform in many applications. They introduce the notion of scale into the analysis. For example, some phenomena being measured may occur at a broader scale, while others appear at a finer scale. Adding this element allows us to zoom in and out of particular features in the data, yet even these methods still rely on the Fourier transform for their design and analysis.

Geometry on the other hand relies in an essential way on a non-Euclidean setting: curves, surfaces, and more generally manifolds. Unfortunately, curved geometry does not readily admit a Fourier transform let alone a fast transform algorithm. Additional complications arise in software and hardware implementations. What used to be simple arrays or streams of regularly arranged data is now a complex topological data structure requiring much more sophisticated algorithms to handle. In general there is no way around these difficulties when the underlying geometry is not flat. For example, consider data defined on a sphere, such as measurements which are a function of direction. One could map the sphere to a section of the plane and then use regular sampling and Fourier transforms. However, it is well known that no such mapping can avoid singularities resulting in uncontrollable artifacts. Instead one needs to take the essential nature of the sphere into account. These arguments apply even more so when the underlying surface is more complicated.

Luckily some recent techniques inspired by wavelet constructions and commonly referred to as "multiresolution algorithms" offer a basis for the development of a *digital geometry processing* toolbox. These techniques, developed at the intersection of mathematics (wavelet analysis) and computer science (graphics), are based on *subdivision*, *lifting*, and *second generation wavelets* and carry over to the curved geometry setting.

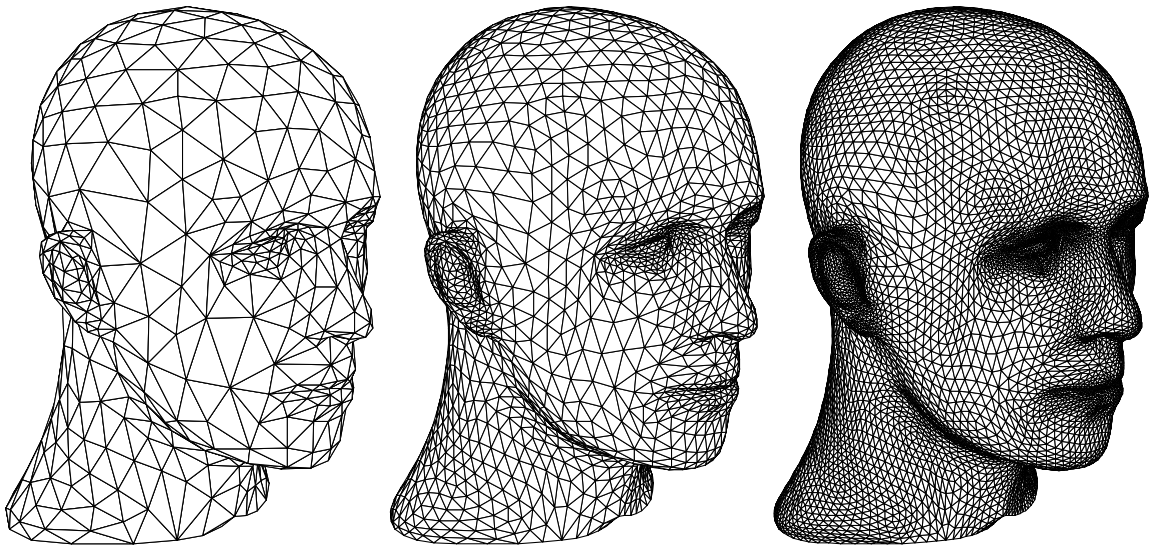


Figure 1: Example of subdivision for a surface, showing 3 successive levels of refinement. On the left an initial triangle mesh approximating the surface. Each triangle is split into 4 according to a particular subdivision rule (middle). On the right the mesh is subdivided in this fashion once again.

The Power of Semi-regular Meshes

We have seen above that it is impossible to keep the regularity, i.e., the regular Cartesian arrangement of samples, of the Euclidean setting when going to surfaces. However, we can do almost as well using so called *semi-regular* meshes. These are constructed through a process of recursive quadrisection, an idea which originated in the area of subdivision. Starting with a coarse mesh consisting of a generally small number of triangles each triangle is recursively split into

[4]In practice algorithms often do not perform an actual Fourier transform to achieve these effects. No less, the Fourier transform is essential to understanding how to design and analyze such algorithms.

four subtriangles. In standard subdivision, where this approach is used to produce smooth surfaces, the new point positions are computed based on local averaging. A few steps in such a sequence are shown in Figure 1. Instead we use point positions which are samples of the desired geometry, in effect adding displacements, or wavelet details, to a subdivision surface. This combination of subdivision and details, or wavelets, can be compared to the Euclidean geometry signal processing setting with its low and high pass filters. While the results from that setting are not immediately applicable to the surface setting, they do provide guidance.

Because of the recursive quadrisection process by which these sampling patterns are built we can build fast hierarchical transforms. These generalize the fast wavelet transform to the surface setting. While it is much harder to prove smoothness and approximation properties in this more general setting first results exists and are extremely encouraging. For example, these constructions can be used for very efficient progressive transmission algorithms. Imagine a file format for geometry which has the property that the first bits in the file provide a rough outline of the shape and as more bits arrive more and more details of the shape are revealed.

Conclusions

The fourth wave of multimedia, which consists of geometry, creates new mathematical and algorithmic challenges which cannot be answered with straightforward extensions of signal processing techniques from the Euclidean setting. However, ideas from multiresolution, subdivision, and second generation wavelets provide the foundation of a new digital geometry processing apparatus based on semi-regular meshes.

Digital Geometry Processing

Peter Schröder
Wim Sweldens

Presenters

Presenters

Brian Curless, Washington university
Igor Guskov, Caltech
Peter Schröder, Caltech
Wim Sweldens, bell labs
Denis Zorin, NYU

Program

Morning

8h30-9h00: Introduction, Wim Sweldens
9h00-10h00: Acquisition, Brian Curless

10h00-10h15: Break

10h15-11h00: Remeshing I, Wim Sweldens
11h00-12h00: Remeshing II, Igor Guskov

Program

Afternoon

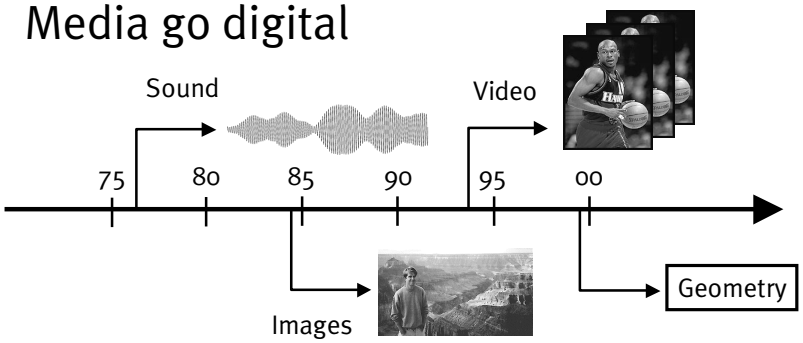
1h30-2h00: Subdivision, Peter Schröder
9h00-10h00: Processing, Denis Zorin

3h00-3h15: Break

3h15-4h00: Compression, Peter Schröder
4h00-5h00: Demos

History of Multimedia

Media go digital

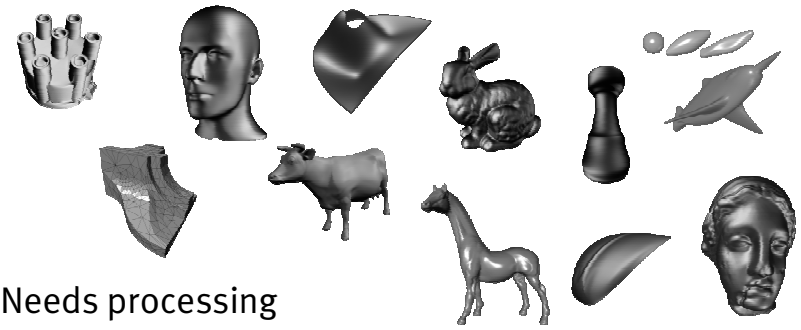


Technology sets off signal processing branches

Digital Geometry

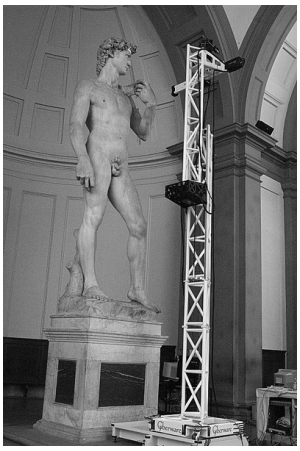
Digitized 3D objects

- 4th wave of digital multimedia



Needs processing

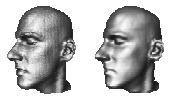
Digital Michelangelo

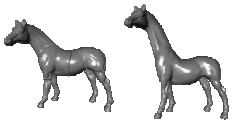


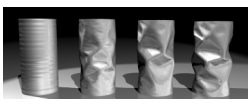
Geometry Processing

Stages

- Creation, acquisition
- Storage, transmission
- Authentication
- Editing, animation
- Simulation
- Manufacture



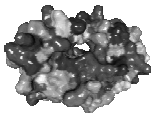




DIGITAL GEOMETRY

Applications

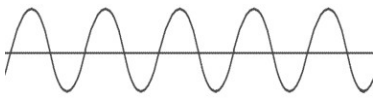
- Cad
- E-catalogs
- Mass customization
- Electronic games
- Medicine & biology
- Art history & archeology


CHALLENGE

Traditional signal processing

- Based on Fourier techniques
  - Decompose in sines and cosines



  - Only for Euclidean, flat geometry
  - Does not easily generalize

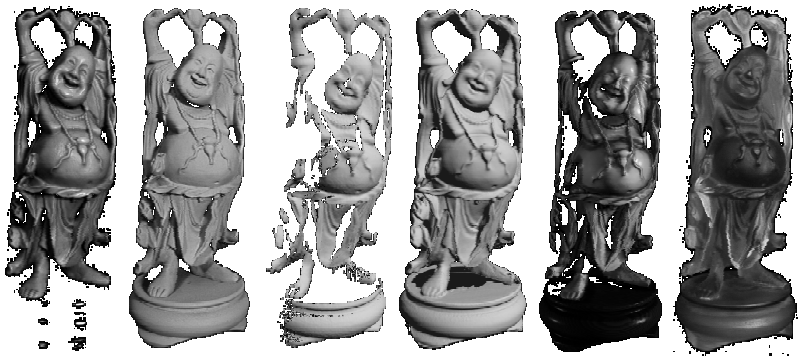
Storyline

Follow stages

- Acquisition
- Remeshing
- Subdivision/details
- Processing
- Compression

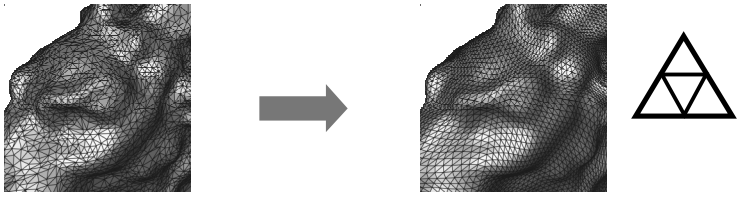
Acquisition: Brian

3D Photography



Remeshing: Wim/Igor

From irregular to regular

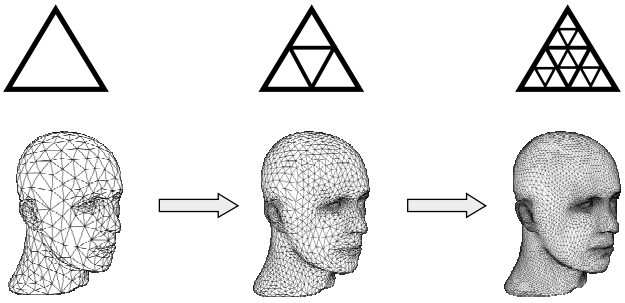


- Easier
  - Datastructure, algorithms
  - Allows subdivision, details

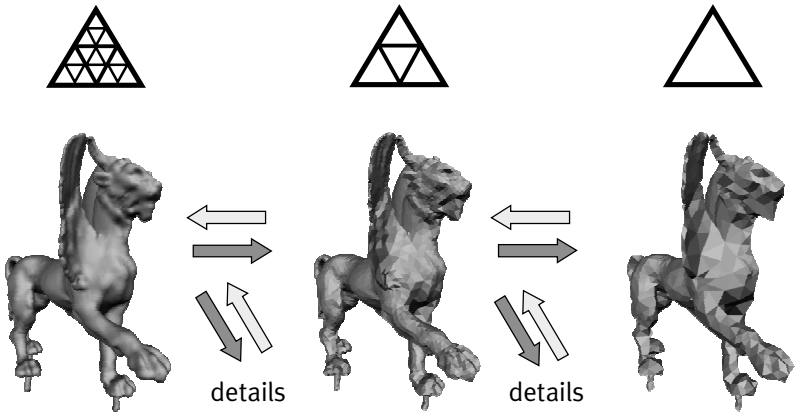
Subdivision: Peter

Create smooth surfaces

- Regular grid

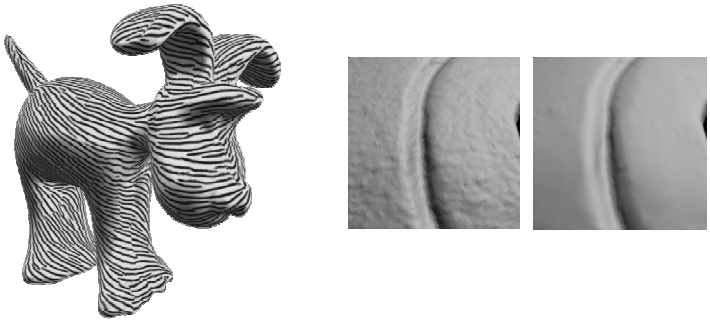


Details/Wavelets



Processing: Denis

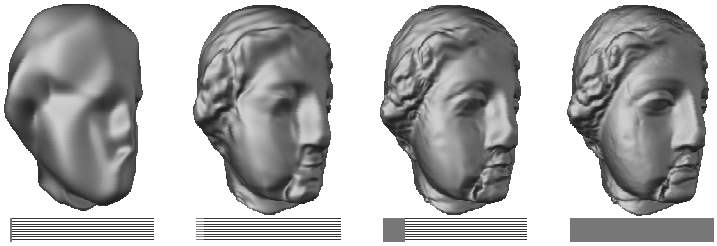
Filtering, texturing



Compression: Peter

Progressive representation

- Based on subdivision/details



476B 1528B 4163B 26800B

SIGGRAPH 01 Course on
Digital Geometry Processing

From range scanners to surfaces

Brian Curless
University of Washington

Overview

Range imaging scanners
- **Imaging radar**
- **Triangulation**

Reconstruction from range images
- **Zippering**
- **Volumetric merging**

Optical range acquisition

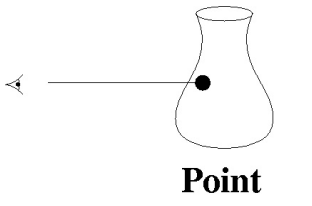
Strengths

- ***Non-contact***
- ***Safe***
- ***Inexpensive (?)***
- ***Fast***

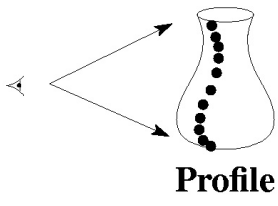
Limitations

- ***Can only acquire visible portions of the surface***
- ***Sensitivity to surface properties***
  > *transparency, shininess, rapid color variations, darkness (no reflected light), subsurface scatter*
- ***Confused by interreflections***

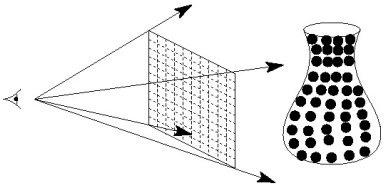
Structure of the data



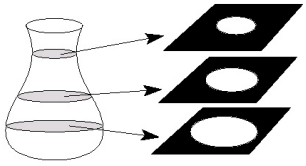
Point



Profile



Range image



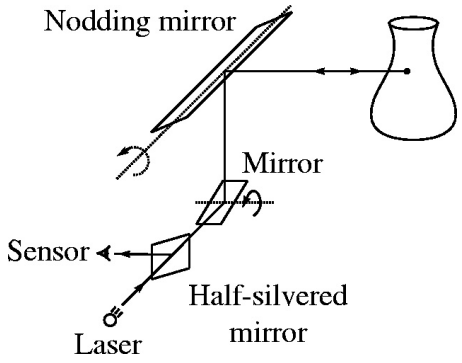
Volumetric

Imaging radar: time of flight

A pulse of light is emitted, and the time of the reflected pulse is recorded:

c t = 2 r = roundtrip distance

Typical scanning configuration:



Imaging radar: Amplitude Modulation

The current to a laser diode is driven at frequency:

$$f_{AM} = \frac{c}{\lambda_{AM}}$$

The phase difference between incoming and outgoing signals gives the range:

$$2r(\Delta\varphi) = n\lambda_{AM} + \frac{\Delta\varphi}{2\pi}\lambda_{AM} \Rightarrow r(\Delta\varphi) = \frac{1}{2}\lambda_{AM}\frac{(\Delta\varphi + 2\pi n)}{2\pi}$$

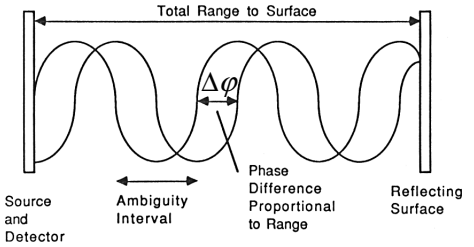


Figure from [Besl'89]

Imaging radar: Amplitude Modulation

Note the ambiguity due to the + 2πn. This translates into range ambiguity:

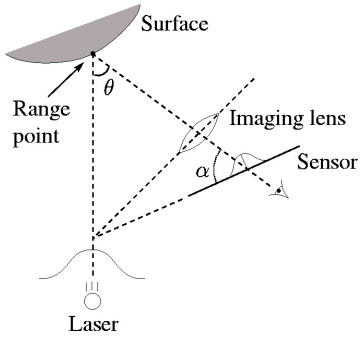
$$r_{ambig} = \frac{n\lambda_{AM}}{2}$$

The ambiguity can be overcome with sweeps of increasingly finer wavelengths.

Optical triangulation

A beam of light strikes the surface, and some of the light bounces toward an off-axis sensor.

The center of the imaged reflection is triangulated against the laser line of sight.



Optical triangulation

Lenses map planes to planes. If the object plane is tilted, then so should the image plane.

The image plane tilt is described by the Scheimpflug condition:

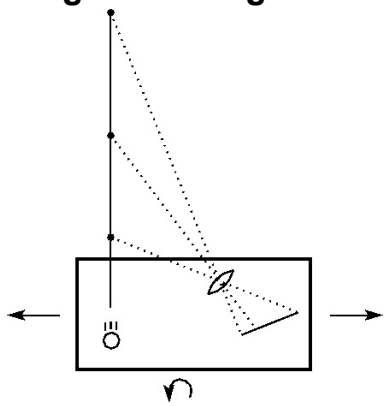
$$\tan\alpha = \frac{\tan\theta}{M}$$

where M is the on-axis magnification.

Triangulation scanning configurations

Moving the laser relative to the camera means loss of focus.

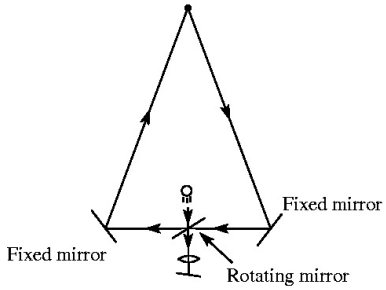
Can instead move the laser and camera together, e.g., by translating or rotating a scanning unit.

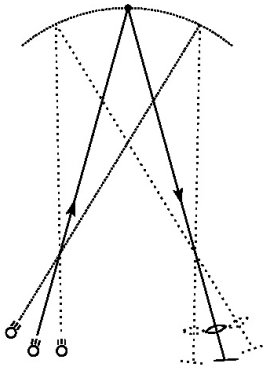


Triangulation scanning configurations

A novel design was created and patented at the NRC of Canada [Rioux'87].

Basic idea: sweep the laser and sensor *simultaneously*.



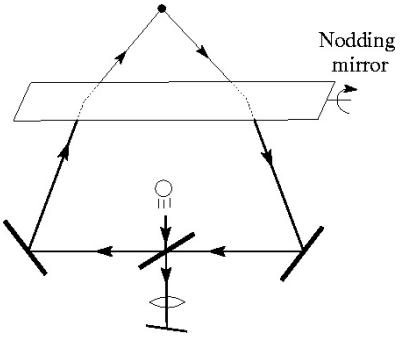


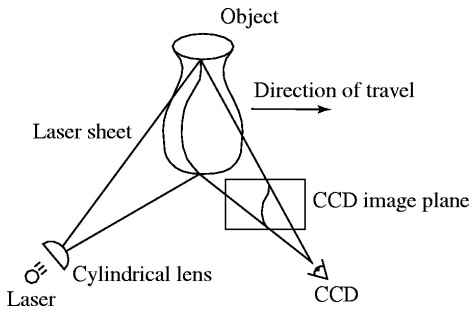
After unfolding the optics

Triangulation scanning configurations

Extension to 3D achievable as:

- ***flying spot***
- ***sweeping light stripe***
- ***hand-held light stripe on jointed arm***



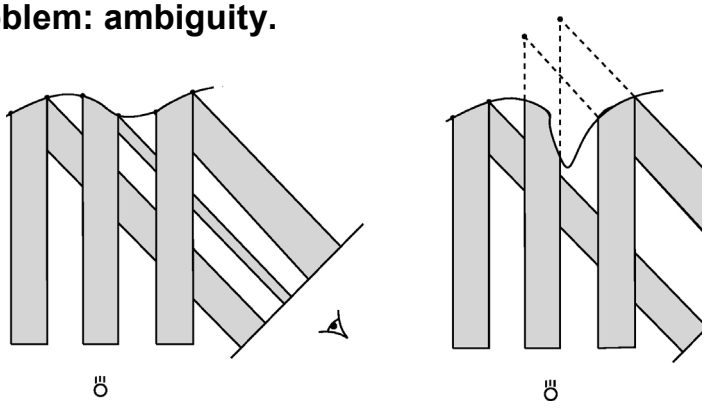


Multi-spot and multi-stripe triangulation

For faster acquisition, some scanners use multiple spots or stripes.

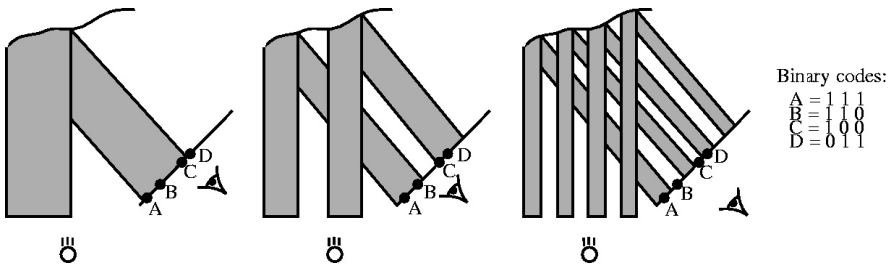
Trade off depth-of-field for speed.

Problem: ambiguity.



Binary coded illumination

Alternative: resolve visibility hierarchically (logN).

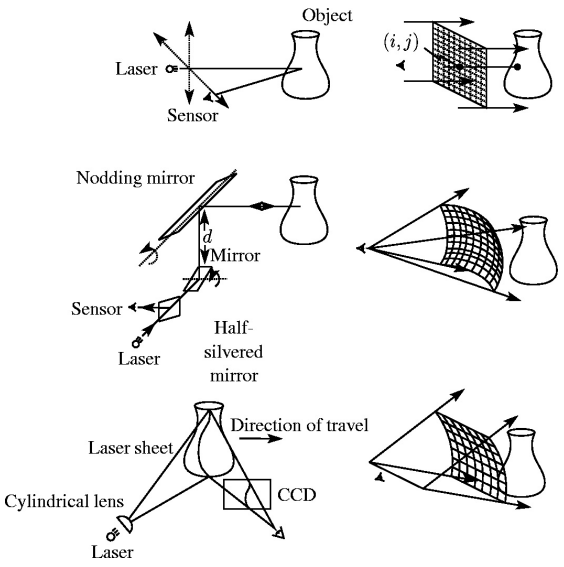


Range images

For many structured light scanners, the range data forms a highly regular pattern known as a *range image*.

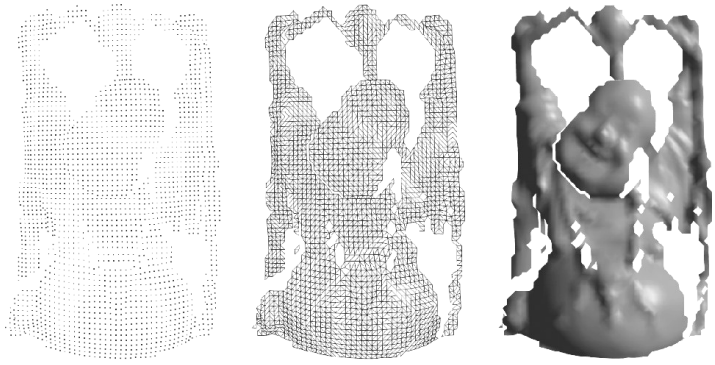
The sampling pattern is determined by the specific scanner.

Examples of sampling patterns



Range images and range surfaces

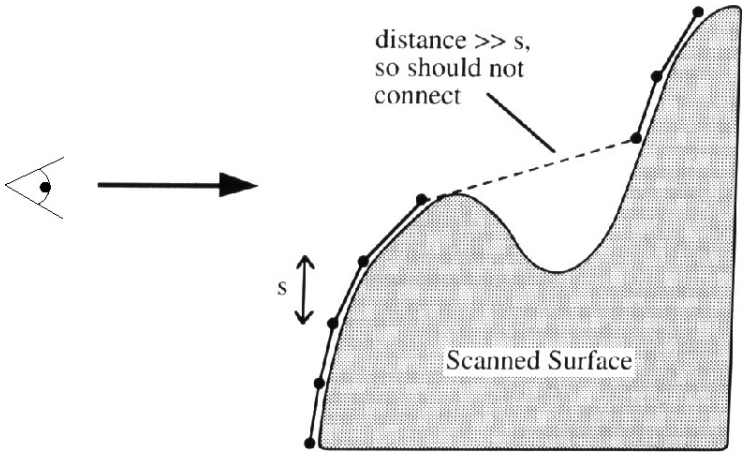
Given a range image, we can perform a preliminary reconstruction known as a *range surface*.



Range image Tesellation Range surface

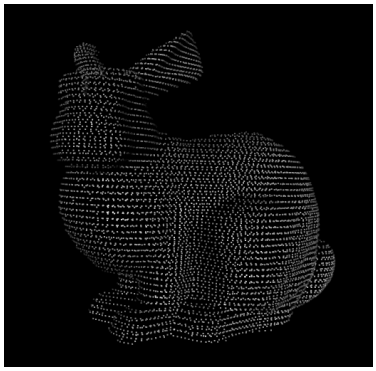
Tessellation threshold

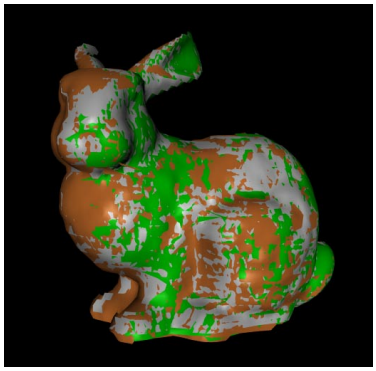
To avoid "prematurely aggressive" reconstruction, a tessellation threshold is employed:



Point clouds vs. range images

We can view the entire set of range data as a point cloud or as a group of overlapping range surfaces.





Surface reconstruction

Given a set of registered range points or images, we want to reconstruct a 2D manifold that closely approximates the surface of the original model.

In this presentation we will focus on two methods for reconstruction from range images:

- ***Zippering***
- ***Volumetric merging***

Desirable properties

Desirable properties for surface reconstruction:

- ***No restriction on topological type***
- ***Representation of range uncertainty***
- ***Utilization of all range data***
- ***Incremental and order independent updating***
- ***Time and space efficiency***
- ***Robustness***
- ***Ability to fill holes in the reconstruction***

Zippering

A number of methods combine range surfaces by stitching polygon meshes together.

Zippering [Turk'94] is one such method.

Overview:

- ***Tessellate range images and assign weights to vertices***
- ***Remove redundant triangles***
- ***Zipper meshes together***
- ***Extract a consensus geometry***

Weight assignment

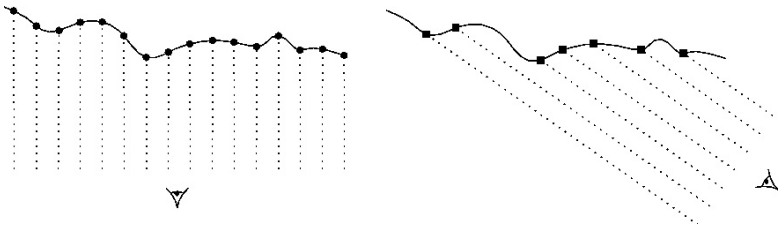
Final surface will be weighted combination of range images.

Weights are assigned at each vertex to:

- ***Favor views with higher sampling rates***
- ***Encourage smooth blends between range images***

Weights for sampling rates

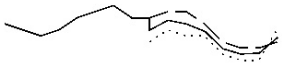
Sampling rate over the surface is highest when view direction is parallel to surface normal.



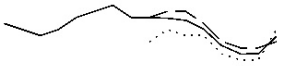
Weights for smooth blends

To assure smooth blends, weights are forced to taper in the vicinity of boundaries:

Two range surfaces

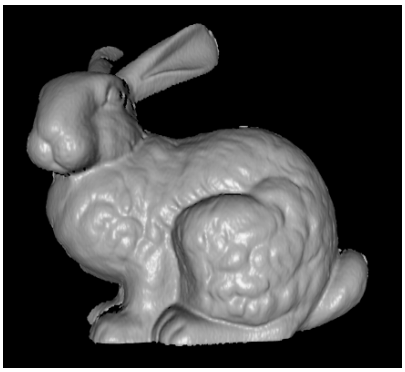


After unweighted blending

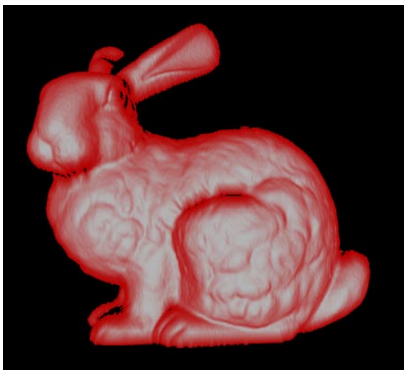


After weighted blending

Example

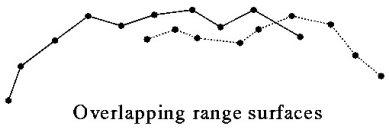


Range surface

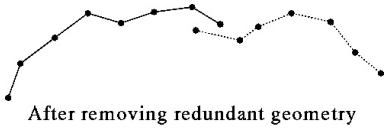


Confidence rendering

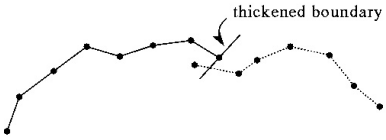
Redundancy removal and zippering



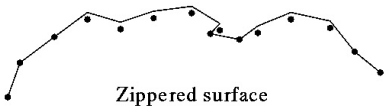
Overlapping range surfaces



After removing redundant geometry

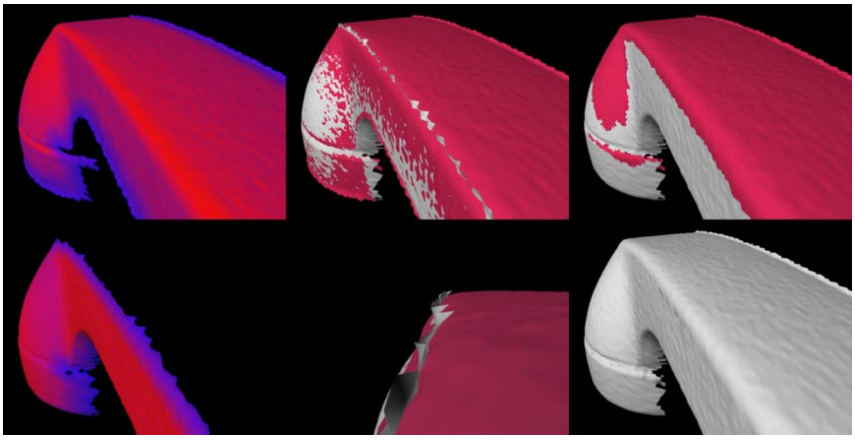


Thickening the mesh boundary for clipping

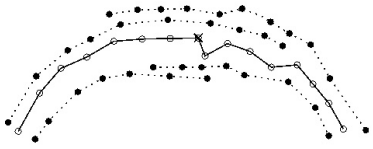


Zippered surface

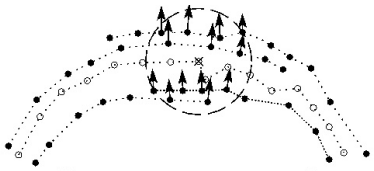
Example



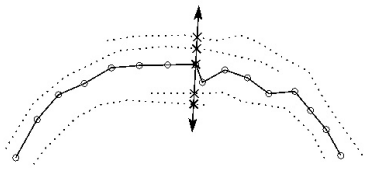
Consensus geometry



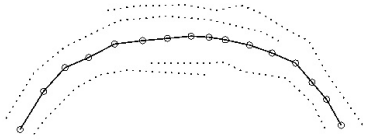
Zippered geometry + range surfaces



Compute consensus normal

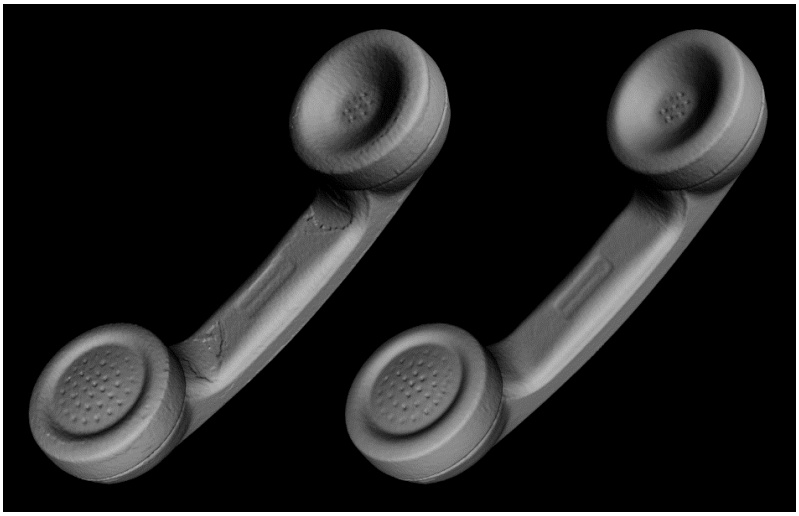


Find vertex positions on range surfaces by intersection with consensus normal



Compute weighted average of vertex positions

Example



Volumetrically combining range images

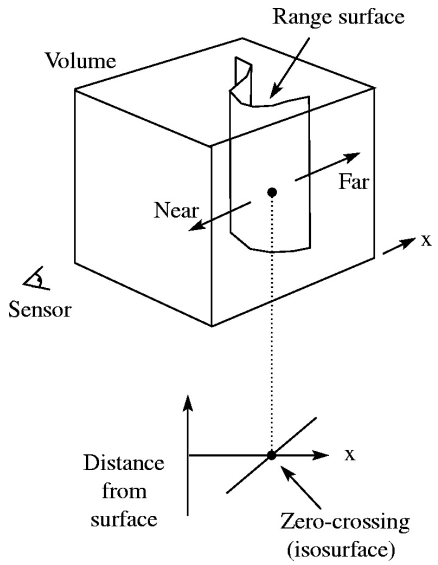
Combining the meshes volumetrically can overcome difficulties of stitching polygon meshes.

Here we describe the method of [Curless'96].

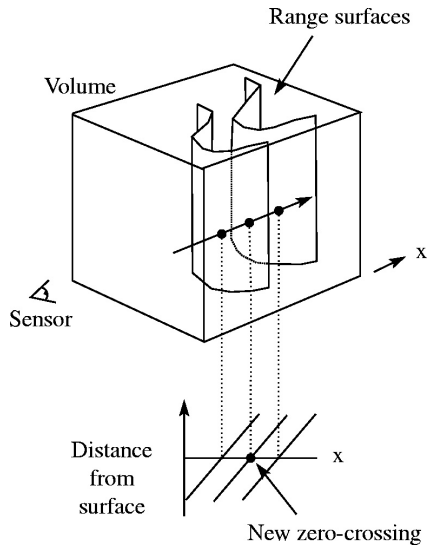
Overview:

- ***Convert range images to signed distance functions***
- ***Combine signed distance functions***
- ***Carve away empty space***
- ***Extract hole-free isosurface***

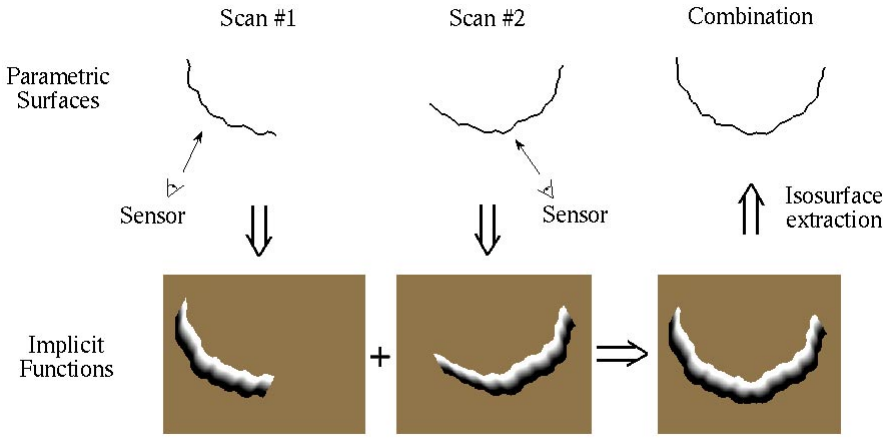
Signed distance function



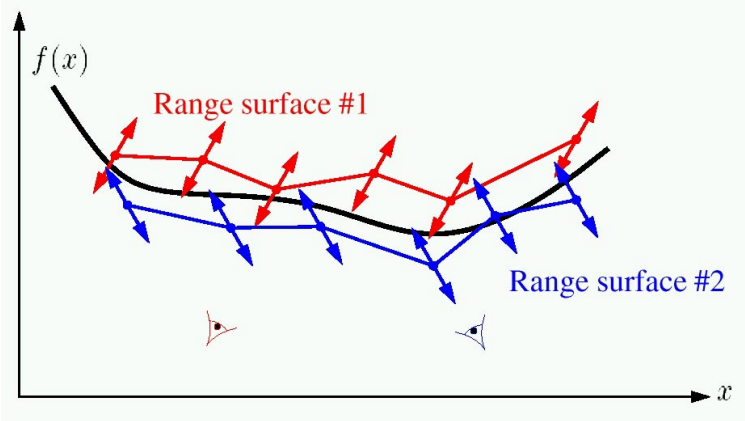
Combining signed distance functions



Merging surfaces in 2D



Least squares solution



Least squares solution

$$E(f) = \sum_{i=1}^{N} \int \overbrace{d_i^2(x, f)dx}^{\text{Error per point}}$$

Error per range surface

Finding the f(x) that minimizes E yields the optimal surface.

This f(x) is exactly the zero-crossing of the combined signed distance functions.

Hole filling

We have presented an algorithm that reconstructs the observed surface. Unseen portions appear as holes in the reconstruction.

A hole-free mesh is useful for:

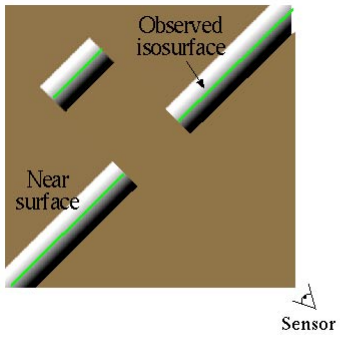
- ***Fitting surfaces to meshes***
- ***Manufacturing models (e.g., stereolithography)***
- ***Aesthetic renderings***

Hole filling

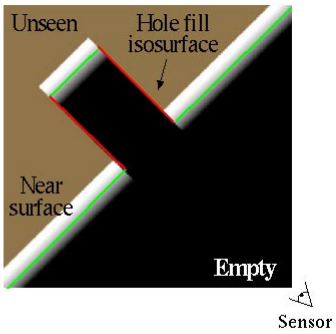
We can fill holes in the polygonal model directly, but such methods:

- ***are hard to make robust***
- ***do not use all available information***

Space carving

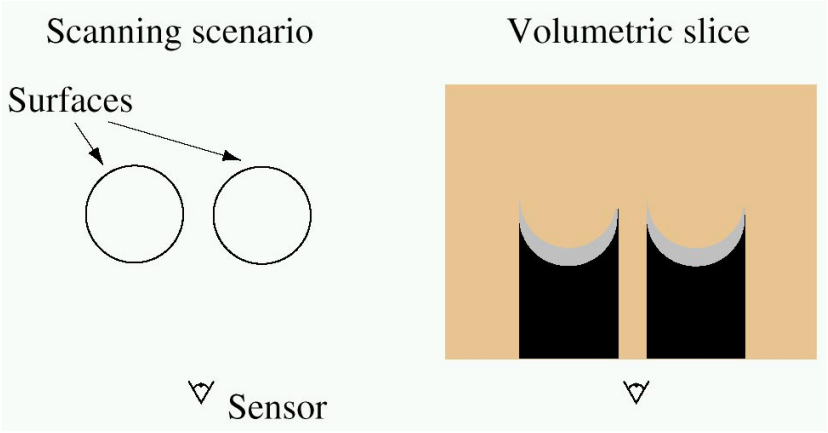


Without space carving

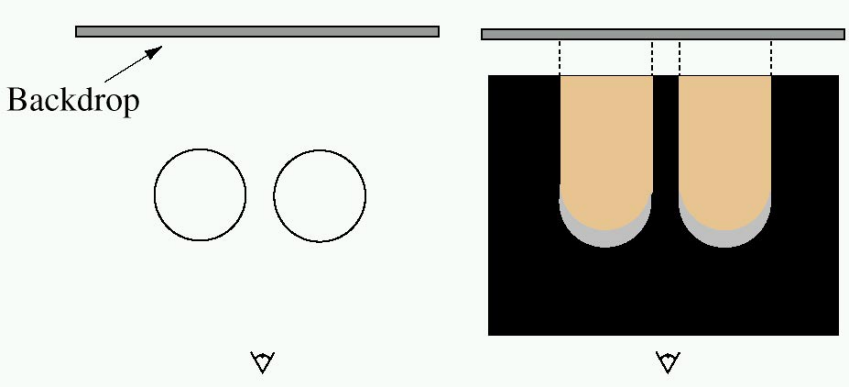


With space carving

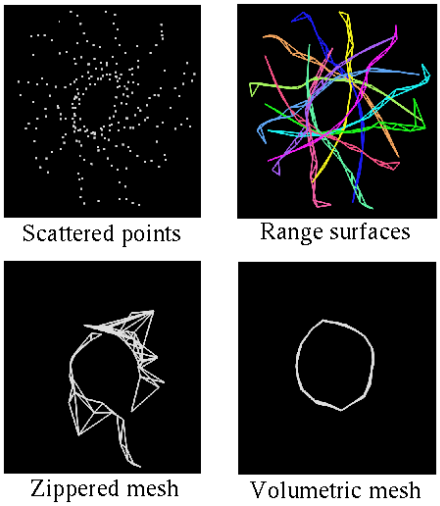
Carving *without* a backdrop



Carving *with* a backdrop



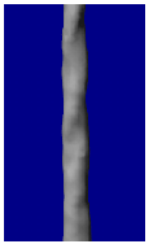
Merging 12 views of a drill bit



Merging 12 views of a drill bit

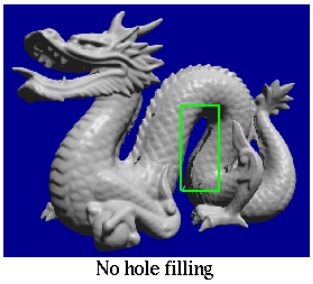
Photograph of painted drill bit

Zippered mesh

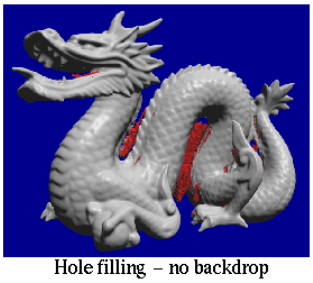


Volumetric mesh

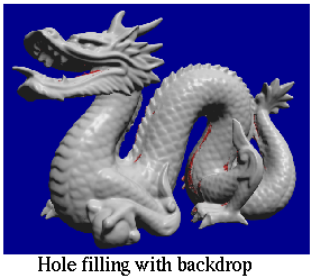
Dragon model



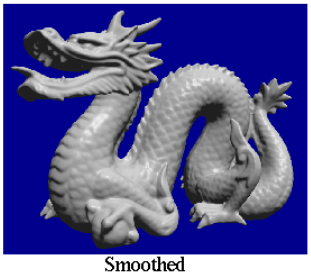
No hole filling



Hole filling – no backdrop

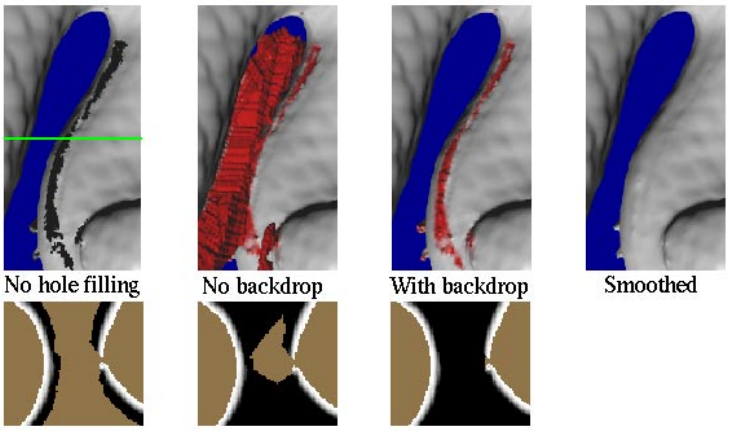


Hole filling with backdrop

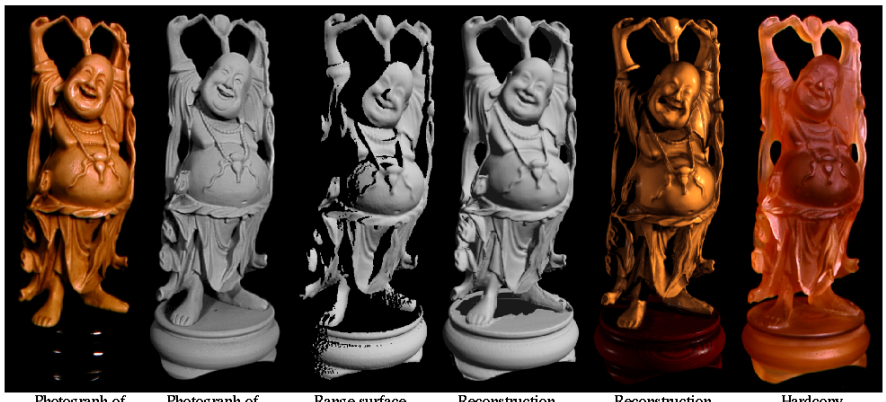


Smoothed

Dragon model



Happy Buddha



Bibliography

Besl, P. *Advances in Machine Vision*. "Chapter 1: Active optical range imaging sensors," pp. 1-63, Springer-Verlag, 1989.

Curless, B. and Levoy, M., "A volumetric method for building complex models from range images." In Proceedings of SIGGRAPH '96, pp. 303-312. ACM Press, August 1996.

Turk, G. and Levoy, M., "Zippered polygon meshes from range images." In Proceedings of SIGGRAPH '94, pp. 311-318. ACM Press, July 1994.

Zippered Polygon Meshes from Range Images

Greg Turk and Marc Levoy
Computer Science Department
Stanford University

Abstract

Range imaging offers an inexpensive and accurate means for digitizing the shape of three-dimensional objects. Because most objects self occlude, no single range image suffices to describe the entire object. We present a method for combining a collection of range images into a single polygonal mesh that completely describes an object to the extent that it is visible from the outside.

The steps in our method are: 1) align the meshes with each other using a modified iterated closest-point algorithm, 2) zipper together adjacent meshes to form a continuous surface that correctly captures the topology of the object, and 3) compute local weighted averages of surface positions on all meshes to form a consensus surface geometry.

Our system differs from previous approaches in that it is incremental; scans are acquired and combined one at a time. This approach allows us to acquire and combine large numbers of scans with minimal storage overhead. Our largest models contain up to 360,000 triangles. All the steps needed to digitize an object that requires up to 10 range scans can be performed using our system with five minutes of user interaction and a few hours of compute time. We show two models created using our method with range data from a commercial rangefinder that employs laser stripe technology.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling.
Additional Key Words: Surface reconstruction, surface fitting, polygon mesh, range images, structured light range scanner.

1 Introduction

This paper presents a method of combining multiple views of an object, captured by a range scanner, and assembling these views into one unbroken polygonal surface. Applications for such a method include:

- Digitizing complex objects for animation and visual simulation.
- Digitizing the shape of a found object such as an archaeological artifact for measurement and for dissemination to the scientific community.
- Digitizing human external anatomy for surgical planning, remote consultation or the compilation of anatomical atlases.
- Digitizing the shape of a damaged machine part to help create a replacement.

E-mail: turk@redclay.stanford.edu, levoy@cs.stanford.edu
Web site: www-graphics.stanford.edu

There is currently no procedure that will allow a user to easily capture a digital description of a physical object. The dream tool would allow one to set an industrial part or a clay figure onto a platform, press a button, and have a complete digital description of that object returned in a few minutes. The reality is that much digitization is done by a user painstakingly touching a 3D sensing probe to hundreds or thousands of positions on the object, then manually specifying the connectivity of these points. Fortunately range scanners offer promise in replacing this tedious operation.

A *range scanner* is any device that senses 3D positions on an object's surface and returns an array of distance values. A *range image* is an $m \times n$ grid of distances (*range points*) that describe a surface either in Cartesian coordinates (a height field) or cylindrical coordinates, with two of the coordinates being implicitly defined by the indices of the grid. Quite a number of measurement techniques can be used to create a range image, including structured light, time-of-flight lasers, radar, sonar, and several methods from the computer vision literature such as depth from stereo, shading, texture, motion and focus. The range images used to create the models in this paper were captured using structured light (described later), but our techniques can be used with any range images where the uncertainties of the distance values are smaller than the spacing between the samples.

Range scanners seem like a natural solution to the problem of capturing a digital description of physical objects. Unfortunately, few objects are simple enough that they can be fully described by a single range image. For instance, a coffee cup handle will obscure a portion of the cup's surface even using a cylindrical scan. To capture the full geometry of a moderately complicated object (e.g. a clay model of a cat) may require as many as a dozen range images.

There are two main issues in creating a single model from multiple range images: *registration* and *integration*. Registration refers to computing a rigid transformation that brings the points of one range image into alignment with the portions of a surface that is shares with another range image. Integration is the process of creating a single surface representation from the sample points from two or more range images.

Our approach to registration uses an iterative process to minimize the distance between two triangle meshes that were created from the range images. We accelerate registration by performing the matching on a hierarchy of increasingly more detailed meshes. This method allows an object to be scanned from any orientation without the need for a six-degree-of-freedom motion device.

We separate the task of integration into two steps: 1) creating a mesh that reflects the topology of the object, and 2) refining the vertex positions of the mesh by averaging the geometric detail that is present in all scans. We capture the topology of an object by merging pairs of triangle meshes that are each created from a single range image. Merging begins by converting two meshes that may have considerable overlap into a pair of meshes that just barely overlap along portions of their boundaries. This is done by simultaneously eating back the boundaries of each mesh that lie directly on top of the other mesh. Next, the meshes are *zippered* together: the triangles of one mesh are clipped to the boundary of the other mesh and the vertices on the boundary are shared. Once all the meshes have been combined, we allow all of the scans to contribute to the surface detail by finding the *consensus geometry*. The final position of a vertex is found by taking an average of nearby positions from each of the original range images. The order in which we perform zippering and consensus geometry is important. We deliberately postpone the refinement of surface geometry until after the overall shape of the object has been determined. This eliminates discontinuities that may be introduced during zippering.

The remainder of this paper is organized as follows. Section 2 describes previous work on combining range images. Section 3 covers the basic principles of a structured light range scanner. Section 4 presents the automatic registration process. Section 5 describes zippering meshes into one continuous surface. Section 6 describes how surface detail is captured through consensus geometry. Section 7 shows examples of digitized models and compares our approach to other methods of combining range data. Section 8 concludes this paper by discussing future work.

2 Previous Work

There is a great deal of published work on registration and integration of depth information, particularly in the vision literature. Our literature review only covers work on registration or integration of dense range data captured by an active range scanner, and where the product of the integration is a polygon mesh.

2.1 Registration

Two themes dominate work in range image registration: matching of "created" features in the images to be matched, and minimization of distances between all points on the surface represented by the two images. In the first category, Wada and co-authors performed six degree of freedom registration by matching distinctive facets from the convex hulls of range images [Wada 93]. They computed a rotation matrix from corresponding facets using a least squares fit of the normal vectors of the facets.

In the second category, Champleboux and co-workers used a data structure called an octree-spline that is a sampled representation of distances to an object's surface [Champleboux 92]. This gave them a rapid way to determine distances from a surface (and the distance gradient) with a low overhead in storage. Chen and Medioni establish a correspondence between points on one surface and nearby tangent planes on the other surface [Chen 92]. They find a rigid motion that minimizes the point-to-tangent collection directly and then iterate. Besl and McKay use an approach they call the *iterated closest-point* algorithm [Besl 92]. This method finds the nearest positions on one surface to a collection of points on the other surface and then transforms one surface so as to minimize the collective distance. They iterate this procedure until convergence.

Our registration method falls into the general category of direct distance minimization algorithms, and is an adaptation of [Besl 92]. It differs in that we do not require that one surface be a strict subset of the other. It is described in Section 4.

2.2 Integration

Integration of multiple range scans can be classified into *structured* and *unstructured* methods. Unstructured integration presumes that one has a procedure that creates a polygonal surface from an arbitrary collection of points in 3-space. Integration in this case is performed by collecting together all the range points from multiple scans and presenting them to the polygonal reconstruction procedure. The Delaunay triangulation of a set of points in 3-space has been proposed as the basis of one such reconstruction method [Boissonnat 84]. Another candidate for surface reconstruction is a generalization of the convex hull of a point set known as the alpha shape [Edelsbrunner 92]. Hoppe and co-authors use graph traversal techniques to help construct a signed distance function from a collection of unorganized points [Hoppe 92]. An isosurface extraction technique produces a polygon mesh from this distance function.

Structured integration methods make use of information about how each point was obtained, such as using error bounds on a point's position or adjacency information between points within one range image. Soucy and Laurendeau use a structured integration technique to combine multiple range images [Soucy 92] that is similar in several respects to our algorithm. Given n range images of an object, they first partition the points into a number of sets that are called *common surface sets*. The range points in one set are then used to create a grid of triangles whose positions are guided by a weighted average of the points in the set. Subsets of these grids are stitched together by a constrained Delaunay triangulation in one of n projections onto a plane. We compare our method to Soucy's in Section 7.

3 Structured Light Range Scanners

In this section we describe the operating principles of range scanners based on structured light. We do this because it highlights issues common to many range scanners and also because the range images used in this article were created by such a scanner.

3.1 Triangulation

Structured light scanners operate on the principle of triangulation (see Figure 1, left). One portion of the scanner projects a specific pattern of light onto the object being scanned. This pattern of light is observed by the sensor of the scanner along a viewing direction that is off-axis from the source of light. The position of the illuminated part of the object is determined by finding the intersection of the light's projected direction and the viewing direction of the sensor. Positions can be accumulated across the length of the object while the object is moved across the path of the projected light. Some of the patterns that have been used in such scanners include a spot, a circle, a line, and several lines at once. Typically the sensor is a CCD array or a lateral effect photodiode.

The scanner used for the examples in this paper is a Cyberware Model 3030 MS. It projects a vertical sheet of He-Ne laser light onto the surface of an object. The laser sheet is created by spreading a laser beam using a cylindrical lens into a sheet roughly 2 mm wide and 30 cm high. The sensor of the Cyberware scanner is a 768×486 pixel CCD array. A typical CCD image shows a ribbon of laser light running from the top to the bottom (see Figure 2). A range point is created by looking across a scanline for the peak intensity of this ribbon. A range point's distance from the scanner (the "depth") is given by the horizontal position of this peak and the vertical position of the range point is given by the number of the scanline. Finding the peaks for each scanline in one frame gives an entire column of range points, and combining the columns from multiple frames as the object is moved through the laser sheet gives the full range image.

3.2 Sources of Error

Any approach to combining range scans should attempt to take into account the possible sources of error inherent in a given scanner. Two sources of error are particularly relevant to integration. One is a result of light falling on the object at a grazing angle. When the projected light falls on a portion of the object that is nearly parallel to the light's path, the sensor sees a dim and stretched-out version of the pattern. Finding the center of the laser sheet when it grazes the

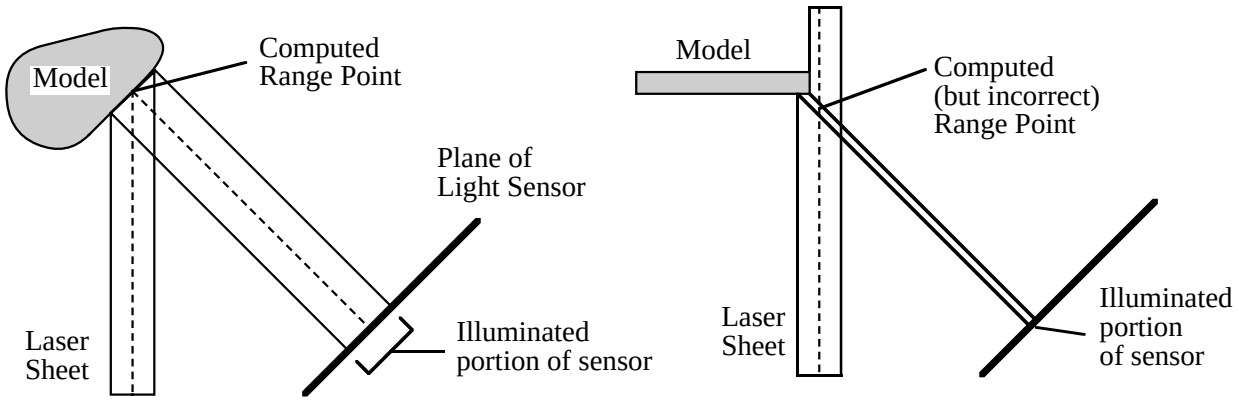


Figure 1: Structured light triangulation (left) and false edge extension in the presence of a partially illuminated edge (right).

object becomes difficult, and this adds uncertainty to the position of the range points. The degree of uncertainty at a given range point can be quantified, and we make use of such information at several stages in our approach to combining range images.

A second source of inaccuracy occurs when only a portion of the laser sheet hits an object, such as when the laser sheet falls off the edge of a book that is perpendicular to the laser sheet (see Figure 1, right). This results in a false position because the peak-detection and triangulation method assumes that the entire width of the sheet is visible. Such an assumption results in edges of objects that are both curled and extended beyond their correct position. This false extension of a surface at edges is an issue that needs to be specifically addressed when combining range images.

3.3 Creating Triangle Meshes from Range Images

We use a mesh of triangles to represent the range image data at all stages of our integration method. Each sample point in the $m \times n$ range image is a potential vertex in the triangle mesh. We take special care to avoid inadvertently joining portions of the surface together that are separated by depth discontinuities (see Figure 3).

To build a mesh, we create zero, one or two triangles from four points of a range image that are in adjacent rows and columns. We find the shortest of the two diagonals between the points and use this to identify the two triplets of points that may become triangles. Each of these point triples is made into a triangle if the edge lengths fall below a distance threshold. Let s be the maximum distance between adjacent range points when we flatten the range image, that is, when we don't include the depth information (see Figure 3). We take the distance threshold be a small multiple of this sampling distance, typically 4s. Although having such a distance threshold may prevent joining some range points that should in fact be connected, we can rely on other range images (those with better views of the location in question) to give the correct adjacency information.

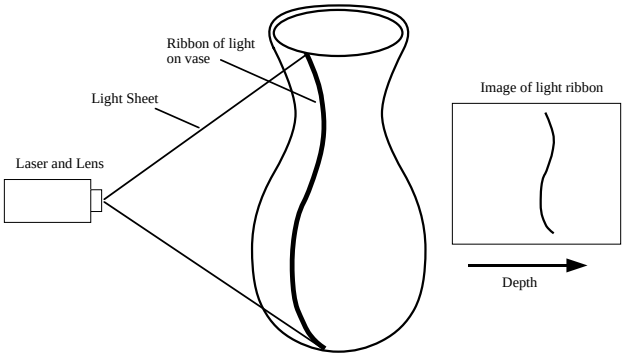


Figure 2: Light-stripe projected on vase (left) and corresponding CCD image (right).

This willingness to discard questionable data is representative of a deliberate overall strategy: to acquire and process large amounts of data rather than draw hypotheses (possibly erroneous) from sparse data. This strategy appears in several places in our algorithm.

4 Registration of Range Images

Once a triangle mesh is created for each range image, we turn to the task of bringing corresponding portions of different range images into alignment with one another. If all range images are captured using a six-degree of freedom precision motion device then the information needed to register them is available from the motion control software. This is the case when the object or scanner is mounted on a robot arm or the motion platform of a precision milling machine. Inexpensive motion platforms are often limited to one or two degrees of freedom, typically translation in a single direction or rotation about an axis. One of our goals is to create an inexpensive system. Consequently, we employ a registration method that does not depend on measured position and orientation. With our scanner, which offers translation and rotation around one axis, we typically take one cylindrical and four translational scans by moving the object with the motion device. To capture the top or the underside of the object, we pick it up by hand and place it on its side. Now the orientation of subsequent scans cannot be matched with those taken earlier, and using a registration method becomes mandatory.

4.1 Iterated Closest-Point Algorithm

This section describes a modified iterated closest-point (ICP) algorithm for quickly registering a pair of meshes created from range images. This method allows a user to crudely align one range image with another on-screen and then invoke an algorithm that snaps the position of one range image into accurate alignment with the other.

The iterated closest-point of [Besl 92] cannot be used to register range images because it requires that *every* point on one surface have

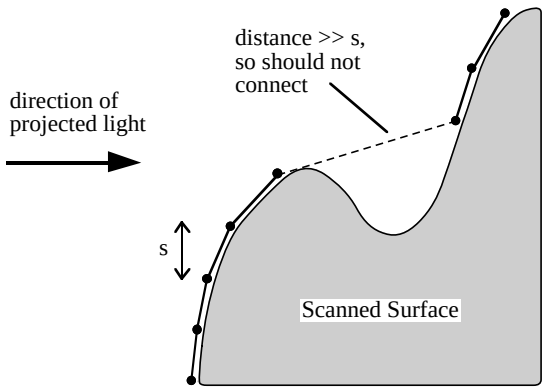


Figure 3: Building triangle mesh from range points.

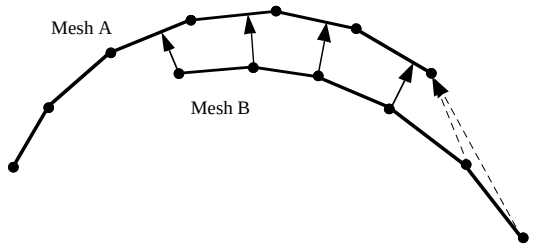


Figure 4: Finding corresponding points for mesh registration. Dotted arrows show matches that should be avoided because they will cause mesh *B* to be erroneously dragged up and left.

a corresponding point on the other surface. Since our scans are overlapping, we seldom produce data that satisfies this requirement. Thus we have developed our own variant of this algorithm. Its steps are:

1) Find the nearest position on mesh *A* to each vertex of mesh *B*.
2) Discard pairs of points that are too far apart.
3) Eliminate pairs in which either points is on a mesh boundary.
4) Find the rigid transformation that minimizes a weighted least-squared distance between the pairs of points.
5) Iterate until convergence.
6) Perform ICP on a more detailed mesh in the hierarchy.

In step 1, it is important to note that we are looking for the 3-space position A_i on the surface of mesh *A* that is closest to a given vertex B_i of mesh *B* (see Figure 4). The nearest point A_i may be a vertex of *A*, may be a point within a triangle, or may lie on a triangle's edge. Allowing these points A_i to be anywhere on a C^0 continuous surface means that the registration between surfaces can have greater accuracy than the spacing *s* between range points.

4.2 Constraints on ICP

Our ICP algorithm differs from Besl's in several ways. First, we have added a distance threshold to the basic iterated closest-point method to avoid matching any vertex B_i of one mesh to a remote part of another mesh that is likely to not correspond to B_i. Such a vertex B_i from mesh *B* might be from a portion of the scanned object that was not captured in the mesh *A*, and thus no pairing should be made to any point on *A*. We have found that excellent registration will result when this distance threshold is set to twice the spacing *s* between range points. Limiting the distance between pairs of corresponding points allows us to perform step 2 (eliminating remote pairs) during the nearest points search in step 1.

The nearest points search can be accelerated considerably by placing the mesh vertices in a uniform subdivision of space based on the distance threshold. Because the triangle size is limited in the mesh creation step, we can search over all triangles within a fixed distance and guarantee that we miss no nearby portion of any triangle. Because we will use this constrained nearest-point search again later, it is worth giving a name to this query. Let nearest_on_mesh(*P*,*d*,*M*) be a routine that returns the nearest position on a mesh *M* to a given point *P*, or that returns nothing if there is no such point within the distance *d*.

Second, we have added the restriction that we never allow boundary points to be part of a match between surfaces. Boundary points are those points that lie on the edge of a triangle and where that edge is not shared by another triangle. Figure 4 illustrates how such matches can drag a mesh in a contrary direction to the majority of the point correspondences.

4.3 Best Rigid Motion

The heart of the iterated closest-point approach is in finding a rigid transformation that minimizes the least-squared distance between the point pairs. Berthold Horn describes a closed-form solution to this problem [Horn 87] that is linear in time with respect to the number of point pairs. Horn's method finds the translation vector *T* and the rotation **R** such that:

$$E = \sum_{i=1}^{n} |A_i - \mathbf{R}(B_i - B_c) - T|^2$$

is minimized, where A_i and B_i are given pairs of positions in 3-space and B_c is the centroid of the B_i. Horn showed that *T* is just the difference between the centroid of the points A_i and the centroid of the points B_i. **R** is found by constructing a cross-covariance matrix between centroid-adjusted pairs of points. The final rotation is given by a unit quaternion that is the eigenvector corresponding to the largest eigenvalue of a matrix constructed from the elements of this cross-covariance matrix. Details can be found in both [Horn 87] and [Besl 92].

As we discussed earlier, not all range points have the same error bounds on their position. We can take advantage of an optional weighting term in Horn's minimization to incorporate the positional uncertainties into the registration process. Let a value in the range from 0 to 1 called *confidence* be a measure of how certain we are of a given range point's position. For the case of structured light scanners, we take the *confidence* of a point *P* on a mesh to be the dot product of the mesh normal *N* at *P* and the vector *L* that points from *P* to the light source of the scanner. (We take the normal at *P* to be the average of the normals of the triangles that meet at *P*.) Additionally, we lower the confidence of vertices near the mesh boundaries to take into account possible error due to false edge extension and curl. We take the confidence of a pair of corresponding points A_i and B_i from two meshes to be the product of their confidences, and we will use w_i to represent this value. The problem is now to find a *weighted* least-squares minimum:

$$E = \sum_{i=1}^{n} w_i |A_i - \mathbf{R}(B_i - B_c) - T|^2$$

The weighted minimization problem is solved in much the same way as before. The translation factor *T* is just the difference between the weighted centroids of the corresponding points. The solution for **R** is described by Horn.

4.4 Alignment in Practice

The above registration method can be made faster by matching increasingly more detailed meshes from a hierarchy. We typically use a mesh hierarchy in which each mesh uses one-forth the number of range points that are used in the next higher level. The less-detailed meshes in this hierarchy are constructed by sub-sampling the range images. Registration begins by running constrained ICP on the lowest-level mesh and then using the resulting transformation as the initial position for the next level up in the hierarchy. The matching distance threshold *d* is halved with each move up the hierarchy.

Besl and McKay describe how to use linear and quadratic extrapolation of the registration parameters to accelerate the alignment process. We use this technique for our alignment at each level in the hierarchy, and find it works well in practice. Details of this method can be found in their paper.

The constrained ICP algorithm registers only two meshes at a time, and there is no obvious extension that will register three or more meshes simultaneously. This is the case with all the registration algorithms we know. If we have meshes *A*, *B*, *C* and *D*, should we register *A* with *B*, then *B* with *C* and finally *C* with *D*, perhaps compounding registration errors? We can minimize this problem by registering all meshes to a single mesh that is created from a cylindrical range image. In this way the cylindrical range image acts as a common anchor for all of the other meshes. Note that if a cylindrical scan covers an object from top to bottom, it captures all the surfaces that lie on the convex hull of the object. This means that,

for almost all objects, there will be some common portions between the cylindrical scan and all linear scans, although the degree of this overlap depends on the extent of the concavities of the object. We used such a cylindrical scan for alignment when constructing the models shown in this paper.

5 Integration: Mesh Zippering

The central step in combining range images is the integration of multiple views into a single model. The goal of integration is to arrive at a description of the overall topology of the object being scanned. In this section we examine how two triangle meshes can be combined into a single surface. The full topology of a surface is realized by zippering new range scans one by one into the final triangle mesh.

Zippering two triangle meshes consists of three steps, each of which we will consider in detail below:

1) Remove overlapping portions of the meshes.
2) Clip one mesh against another.
3) Remove the small triangles introduced during clipping.

5.1 Removing Redundant Surfaces

Before attempting to join a pair of meshes, we eat away at the boundaries of both meshes until they just meet. We remove those triangles in each mesh that are in some sense "redundant," in that the other mesh includes an unbroken surface at that same position in space. Although this step removes triangles from the meshes, we are not discarding data since all range points eventually will be used to find the consensus geometry (Section 6). Given two triangle meshes *A* and *B*, here is the process that removes their redundant portions:

Repeat until both meshes remain unchanged:
  Remove redundant triangles on the boundary of mesh *A*
  Remove redundant triangles on the boundary of mesh *B*

Before we can remove a given triangle *T* from mesh *A*, we need to determine whether the triangle is redundant. We accomplish this by querying mesh *B* using the nearest_on_mesh() routine that was introduced earlier. In particular, we ask for the nearest positions on mesh *B* to the vertices V_1, V_2 and V_3 of *T*. We will declare *T* to be redundant if the three queries return positions on *B* that are within a tolerance distance *d* and if none of these positions are on the boundary of *B*. Figure 7 shows two overlapping surfaces before and after removing their redundant triangles. In some cases this particular decision procedure for removing triangles will leave tiny gaps where the meshes meet. The resulting holes are no larger than the maximum triangle size and we currently fill them in an automatic post-processing step to zippering. Using the fast triangle redundancy check was an implementation decision for the sake of efficiency, not a necessary characteristic of our zippering approach, and it could easily be replaced by a more cautious redundancy check that leaves no gaps. We have not found this necessary in practice.

If we have a measure of confidence of the vertex positions (as we do for structured light scanners), then the above method can be altered to preserve the more confident vertices. When checking to see if the vertices V_1, V_2 and V_3 of *T* lie within the distance tolerance of mesh *B*, we also determine whether at least two of these vertices have a lower confidence measure than the nearby points on *B*. If this is the case, we allow the triangle to be removed. When no more triangles can be removed from the boundaries of either mesh, we drop this confidence value restriction and continue the process until no more changes can be made. This procedure results in a pair of meshes that meet along boundaries of nearly equal confidences.

5.2 Mesh Clipping

We now describe how triangle clipping can be used to smoothly join two meshes that slightly overlap. The left portion of Figure 5 shows two overlapping meshes and the right portion shows the result of clipping. Let us examine the clipping process in greater detail, and for the time being make the assumption that we are operating on two meshes that lie in a common plane.

To clip mesh *A* against the boundary of mesh *B* we first need to add new vertices to the boundary of *B*. Specifically, we place a new vertex wherever an edge of a triangle from mesh *A* intersects the boundary of mesh *B*. Let *Q* be the set of all such new vertices. Together, the new vertices in *Q* and the old boundary vertices of mesh *B* will form a common boundary that the triangles from both meshes will share. Once this new boundary is formed we need to incorporate the vertices *Q* into the triangles that share this boundary. Triangles from mesh *B* need only to be split once for each new vertex to be incorporated (shown in Figure 5, right). Then we need to divide each border triangle from *A* into two parts, one part that lies inside the boundary of *B* that should be discarded and the other part that lies outside of this boundary and should be retained (See Figure 5, middle). The vertices of the retained portions of the triangle are passed to a constrained triangulation routine that returns a set of triangles that incorporates all the necessary vertices (Figure 5, right).

The only modification needed to extend this clipping step to 3-space is to determine precisely how to find the points of intersection *Q*. In 3-space the edges of mesh *A* might very well pass above or below the boundary of *B* instead of exactly intersecting the boundary. To correct for this we "thicken" the boundary of mesh *B*. In essence we create a wall that runs around the boundary of *B* and that is roughly perpendicular to *B* at any given location along the boundary. The portion of the wall at any given edge *E* is a collection of four triangles, as shown in Figure 6. To find the intersection points with the edges of *A*, we only need to note where these edges pass through the wall of triangles. We then move this intersection point down to the nearest position on the edge *E* to which the intersected portion of the wall belongs. The rest of the clipping can proceed as described above.

5.3 Removing Small Triangles

The clipping process can introduce arbitrarily small or thin triangles into a mesh. For many applications this does matter, but in situations where such triangles are undesirable they can easily be removed. We use vertex deletion to remove small triangles: if any of a triangle's altitudes fall below a user-specified threshold we delete one of the triangle's vertices and all the triangles that shared this vertex. We then use constrained triangulation to fill the hole that is left by deleting these triangles (see [Bern 92]). We preferentially delete vertices that were introduced as new vertices during the clipping process. If all of a triangle's vertices are original range points then the vertex opposite the longest side is deleted.

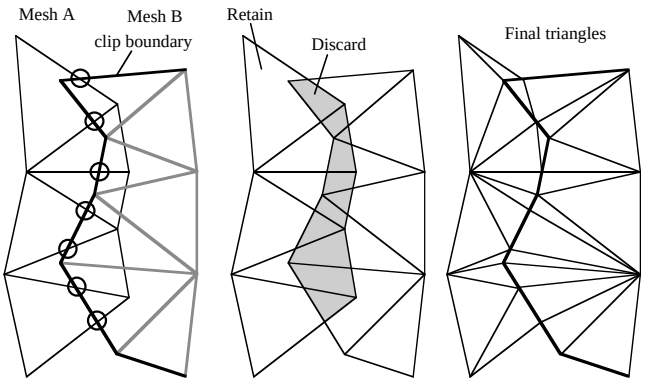


Figure 5: Mesh *A* is clipped against the boundary of mesh *B*. Circles (left) show intersection between edges of *A* and *B*'s boundary. Portions of triangles from *A* are discarded (middle) and then both meshes incorporate the points of intersection (right).

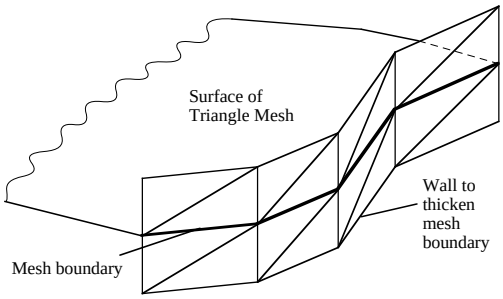


Figure 6: Thickened boundary for clipping in 3-space.

5.4 False Edge Extension

As described in Section 3.2, range points from a structured light scanner that are near an object's silhouette are extended and curled away from the true geometry. These extended edges typically occur at corners. If there is at least one scan that spans both sides of the corner, then our method will correctly reconstruct the surface at the corner. Since we lower the confidence of a surface near the mesh boundaries, triangles at the false edge extensions will be eliminated during redundant surface removal because there are nearby triangles with higher confidence in the scan that spans the corner. For correct integration at a corner, it is the user's responsibility to provide a scan that spans both sides of the corner. Figure 7 illustrates correct integration at a corner in the presence of false edge extension. Unfortunately, no disambiguating scan can be found when an object is highly curved such as a thin cylinder.

Although the problem of false edge extension is discussed in the structured light literature [Businski 92], we know of no paper on surface integration from such range images that addresses or even mentions this issue. We are also unaware of any other integration methods that will correctly determine the geometry of a surface at locations where there are false extensions. Our group has developed a method of reducing false edge extensions when creating the range images (to appear in a forthcoming paper) and we are exploring algorithms that will lessen the effect of such errors during integration. It is our hope that by emphasizing this issue we will encourage others to address this topic in future research on range image integration.

6 Consensus Geometry

When we have zippered the meshes of all the range images together, the resulting triangle mesh captures the *topology* of the scanned object. This mesh may be sufficient for some applications. If surface detail is important, however, we need to fine-tune the *geometry* of the mesh.

The final model of an object should incorporate all the information available about surface detail from each range image of the object. Some of this information may have been discarded when we removed redundant triangles during mesh zippering. We re-introduce the information about surface detail by moving each vertex of our zippered mesh to a consensus position given by a weighted average of positions from the original range images. Vertices are moved only in the direction of the surface normal so that features are not blurred by lateral motion. This is in contrast to unstructured techniques which tend to blur small features isotropically. Our preference for averaging only in the direction of the surface normal is based on the observation that most points in range scans are generally accurately placed with respect to other points in the same scan, but may differ between scans due to alignment errors such as uncorrected optical distortion in the camera. Let M_1, M_2,..., M_n refer to the original triangle meshes created from the range images. Then the three steps for finding the consensus surface are:

1) Find a local approximation to the surface normal.
2) Intersect a line oriented along this normal with each original range image.
3) Form a weighted average of the points of intersection.

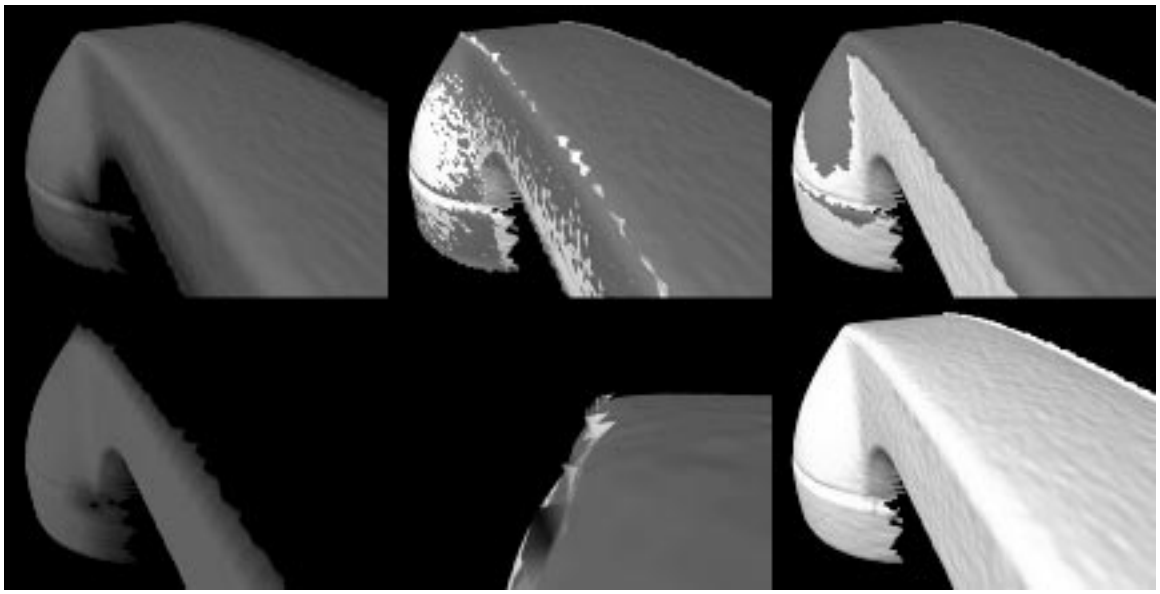


Figure 7: Left (top and bottom): Meshes created from two range images of a telephone. Red denotes locations of high confidence and blue shows low confidence. Note the low confidence at the edges to account for false edge extensions. Top middle: The two meshes (colored red and white) after alignment. Bottom middle: Close-up of aligned meshes that shows a jagged ridge of triangles that is the false edge extension of the white mesh at a corner. Top right: The meshes after redundant surface removal. Bottom right: The meshes after zippering.

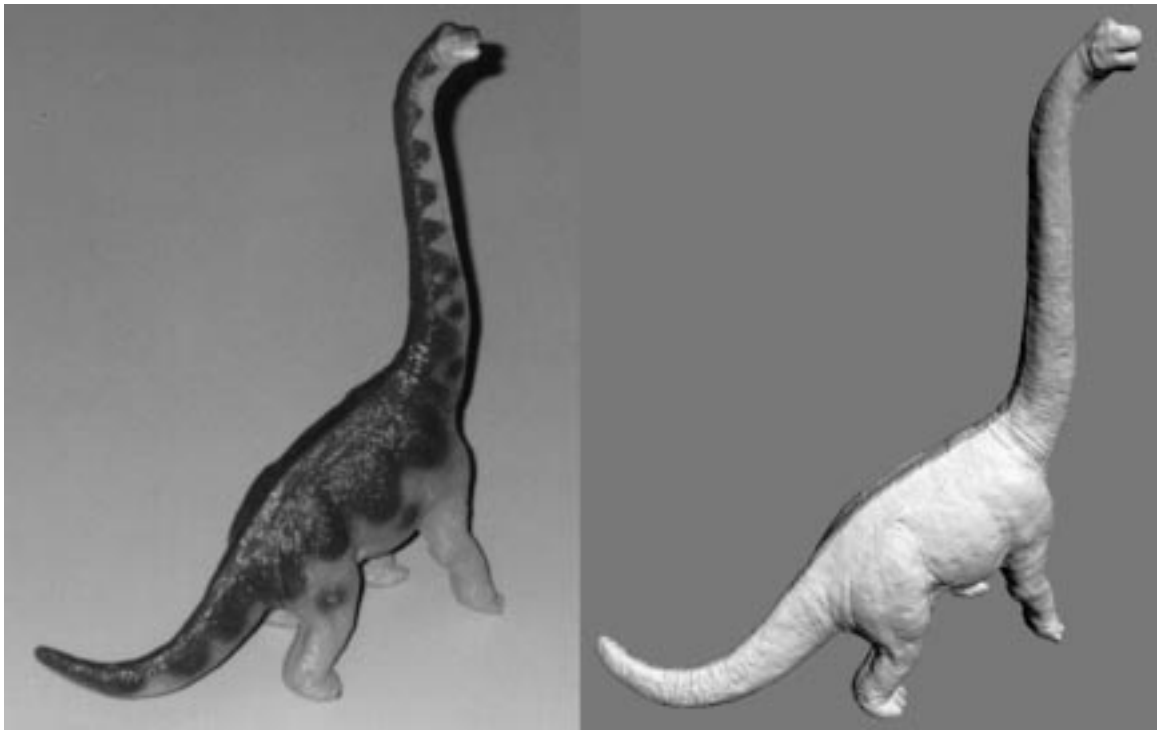


Figure 8: Photograph of a plastic dinosaur model (left) and a polygon mesh created by registering and zippering together 14 range images that were taken of the model (right). The mesh consists of more than 360,000 polygons.

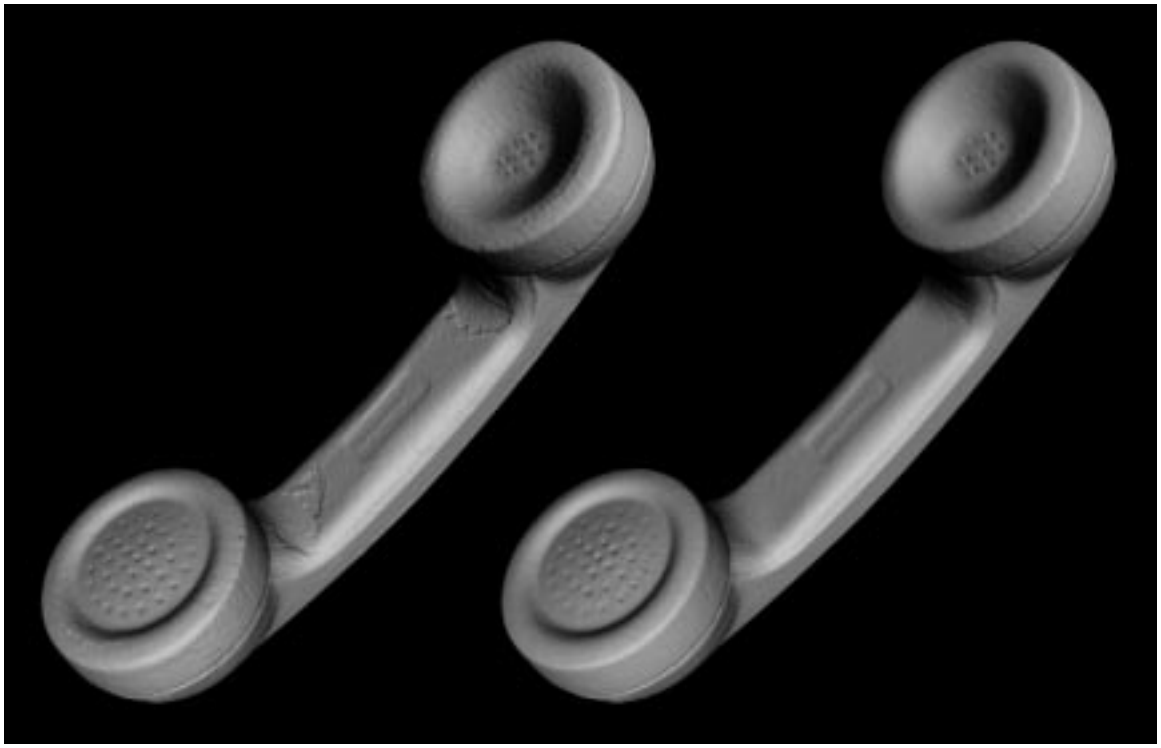


Figure 9: Left: This model of a telephone handset was created by zippering together meshes from ten range images. The mesh consists of more than 160,000 triangles. Right: The final positions of the vertices in the mesh have been moved to an average of nearby positions in the original range images. We call this the consensus geometry.

We approximate the surface normal N at a given vertex V by taking an average over all vertex normals from the vertices in all the meshes M_i that fall within a small sphere centered at V. We then intersect each of the meshes M_i with the line passing through V along the direction N. Let P be the set of all intersections that are near V. We take the consensus position of V to be the average of all the points in P. If we have a measure of confidence for positions on a mesh we use this to weight the average.

7 Results and Discussion

The dinosaur model shown in Figure 8 was created from 14 range images and contains more than 360,000 triangles. Our integration method correctly joined together the meshes at all locations except on the head where some holes due to false edge extensions were filled manually. Such holes should not occur once we eliminate the false extensions in the range images. The dinosaur model was assembled from a larger quantity of range data (measured either in number of scans or number of range points) than any published model known to us. Naturally, we plan to explore the use of automatic simplification methods with our models [Schroeder 92] [Turk 92] [Hoppe 93]. Figure 9 shows a model of a phone that was created from ten range images and contains over 160,000 triangles. The mesh on the right demonstrates that the consensus geometry both reduces noise from the range images without blurring the model's features and also that it eliminates discontinuities at zippered regions.

A key factor that distinguishes our approach from those using unstructured integration ([Hoppe 92] and others) is that our method attempts to retain as much of the triangle connectivity as is possible from the meshes created from the original range images. Our integration process concentrates on a one-dimensional portion of the mesh (the boundary) instead of across an entire two-dimensional surface, and this makes for rapid integration.

Our algorithm shares several characteristics with the approach of Soucy and Laurendeau, which is also a structured integration method [Soucy 92]. The most important difference is the order in which the two methods perform integration and geometry averaging. Soucy's method first creates the final vertex positions by averaging between range images and then stitches together the common surface sets. By determining geometry before connectivity, their approach may be sensitive to artifacts of the stitching process. This is particularly undesirable because their method can create seams between as many as 2^n common surface sets from n range images. Such artifacts are minimized in our approach by performing geometry averaging after zippering.

In summary, we use zippering of triangle meshes followed by refinement of surface geometry to build detailed models from range scans. We expect that in the near future range image technology will replace manual digitization of models in several application areas.

8 Future Work

There are several open problems related to integration of multiple range images. One issue is how an algorithm might automatically determine the next best view to capture more of an object's surface. Another important issue is merging reflectance information (including color) with the geometry of an object. Maybe the biggest outstanding issue is how to create higher-order surface descriptions such as Bezier patches or NURBS from range data, perhaps guided by a polygon model.

Acknowledgments

We thank David Addleman, George Dabrowski and all the other people at Cyberware for the use of a scanner and for educating us about the issues involved in the technology. We thank all the members of our scanner group for numerous helpful discussions. In particular, Brian Curless provided some key insights for interpreting the range data and also wrote code to help this work. Thanks to Phil Lacroute for help with the color figures. This work was supported by an IBM Faculty Development Award, The Powell Foundation, and the National Science Foundation under contract CCR-9157767.

References

[Bern 92] Bern, Marshall and David Eppstein, "Mesh Generation and Optimal Triangulation," Technical Report P92-00047, Xerox Palo Alto Research Center, March 1992.

[Besl 92] Besl, Paul J. and Neil D. McKay, "A Method of Registration of 3-D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14, No. 2 (February 1992), pp. 239–256.

[Boissonnat 84] Boissonnat, Jean-Daniel, "Geometric Structures for Three-Dimensional Shape Representation," *ACM Transactions on Graphics*, Vol. 3, No. 4 (October 1984), pp. 266–286.

[Businski 92] Businski, M., A. Levine and W. H. Stevenson, "Performance Characteristics of Range Sensors Utilizing Optical Triangulation," *IEEE National Aerospace and Electronics Conference*, Vol. 3 (1992), pp. 1230–1236.

[Champleboux 92] Champleboux, Guillaume, Stephane Lavallee, Richard Szeliski and Lionel Brunie, "From Accurate Range Imaging Sensor Calibration to Accurate Model-Based 3-D Object Localization," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Champaign, Illinois, June 15-20, 1992, pp. 83–89.

[Chen 92] Chen, Yang and Gerard Medioni, "Object Modelling by Registration of Multiple Range Images," *Image and Vision Computing*, Vol. 10, No. 3 (April 1992), pp. 145–155.

[Edelsbrunner 92] Edelsbrunner, Herbert and Ernst P. Mücke, "Three-dimensional Alpha Shapes," *Proceedings of the 1992 Workshop on Volume Visualization*, Boston, October 19-20, 1992, pp. 75–82.

[Hoppe 92] Hoppe, Hugues, Tony DeRose, Tom Duchamp, John McDonald and Werner Stuetzle, "Surface Reconstruction from Unorganized Points," *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH '92), pp. 71–78.

[Hoppe 93] Hoppe, Hugues, Tony DeRose, Tom Duchamp, John McDonald and Werner Stuetzle, "Mesh Optimization," *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH '93), pp. 19–26.

[Horn 87] Horn, Berthold K. P., "Closed-Form Solution of Absolute Orientation Using Unit Quaternions," *Journal of the Optical Society of America. A*, Vol. 4, No. 4 (April 1987), pp. 629–642.

[Schroeder 92] Schroeder, William J., Jonathan A. Zarge and William E. Lorensen, "Decimation of Triangle Meshes," *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH '92), pp. 65–70.

[Soucy 92] Soucy, Marc and Denis Laurendeau, "Multi-Resolution Surface Modeling from Multiple Range Views," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Champaign, Illinois, June 15-20, 1992, pp. 348–353.

[Turk 92] Turk, Greg, "Re-Tiling Polygonal Surfaces," *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH '92), pp. 55–64.

[Wada 93] Wada, Nobuhiko, Hiroshi Toriyama, Hiromi T. Tanaka and Fumio Kishino, "Reconstruction of an Object Shape from Multiple Incomplete Range Data Sets Using Convex Hulls," *Computer Graphics International '93*, Lausanne, Switzerland, June 21-25, 1993, pp. 193–203.

Reprinted, with permission, from IEEE Int't Conference on Computer Vision '95, pp. 987-994, June, 1995.

Better Optical Triangulation through Spacetime Analysis

Brian Curless

curless@cs.stanford.edu
Computer Systems Laboratory
Stanford University
Stanford, CA 94305

Marc Levoy

levoy@cs.stanford.edu
Computer Science Department
Stanford University
Stanford, CA 94305

Abstract

The standard methods for extracting range data from optical triangulation scanners are accurate only for planar objects of uniform reflectance illuminated by an incoherent source. Using these methods, curved surfaces, discontinuous surfaces, and surfaces of varying reflectance cause systematic distortions of the range data. Coherent light sources such as lasers introduce speckle artifacts that further degrade the data. We present a new ranging method based on analyzing the time evolution of the structured light reflections. Using our spacetime analysis, we can correct for each of these artifacts, thereby attaining significantly higher accuracy using existing technology. We present results that demonstrate the validity of our method using a commercial laser stripe triangulation scanner.

1 Introduction

Active optical triangulation is one of the most common methods for acquiring range data. Although this technology has been in use for over twenty years, its speed and accuracy has increased dramatically in recent years with the development of geometrically stable imaging sensors such as CCD's and lateral effect photodiodes. The range acquisition literature contains many descriptions of optical triangulation range scanners, of which we list a handful [2] [8] [10] [12] [14] [17] . The variety of methods differ primarily in the structure of the illuminant (typically point, stripe, multi-point, or multi-stripe), the dimensionality of the sensor (linear array or CCD grid), and the scanning method (move the object or move the scanner hardware).

Figure 1 shows a typical system configuration in two dimensions. The location of the center of the reflected light pulse imaged on the sensor corresponds to a line of sight that intersects the illuminant in exactly one point, yielding a depth value. The shape of the object is acquired by translating or rotating the object through the beam or by scanning the beam across the object.

The accuracy of optical triangulation methods hinges on the ability to locate the "center" of the imaged pulse at each time step. For optical triangulation systems that extract range from single imaged pulses at a time, variations in surface reflectance and shape result in systematic range errors. Several researchers have observed one or both of these accuracy limitations [4] [12] [16]. For the case of coherent illumination, the images of reflections from rough surfaces are also subject to laser speckle noise, introducing noise into the range data. Researchers have studied the effect of speckle on range determination and have indicated that it is a fundamental limit to the accuracy of laser range triangulation, though its effects can be reduced with well-known speckle reduction techniques [1] [5]. Mundy and Porter [12] attempt to correct for variations in surface reflectance by noting that two imaged pulses, differing in position or wavelength are sufficient to overcome the reflectance errors, though some restrictive assumptions are necessary for the case of differing wavelengths. Kanade, et al, [11] describe a rangefinder that finds peaks in time for a stationary sensor with pixels that view fixed points on an object. This method of peak detection is very similar to the one presented in this paper for solving some of the problems of optical triangulation; however, the authors in [11] do not indicate that their design solves or even addresses these problems. Further, we show that the principle generalizes to other scanning geometries.

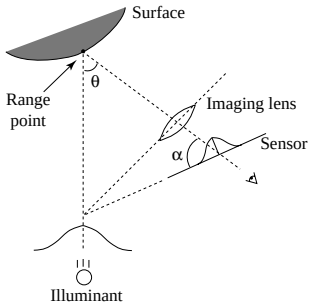


Figure 1: Optical triangulation geometry. The angle θ is the triangulation angle while α is the tilt of the sensor plane needed to keep the laser plane in focus.

In the following sections, we first show how range errors arise with traditional triangulation techniques. In section 3, we show that by analyzing the time evolution of structured light reflections, a process we call spacetime analysis, we can overcome the accuracy limitations caused by shape and reflectance variations. Experimental evidence also indicates that laser speckle behaves in a manner that allows us to reduce its distorting effect as well.

In sections 4 and 5, we describe our hardware and software implementation of the spacetime analysis using a commercial scanner and a video digitizer, and we demonstrate a significant improvement in range accuracy. Finally, in section 6, we conclude and describe future directions.

2 Error in triangulation systems

For optical triangulation systems, the accuracy of the range data depends on proper interpretation of imaged light reflections. The most common approach is to reduce the problem to one of finding the "center" of a one dimensional pulse, where the "center" refers to the position on the sensor which hopefully maps to the center of the illuminant. Typically, researchers have opted for a statistic such

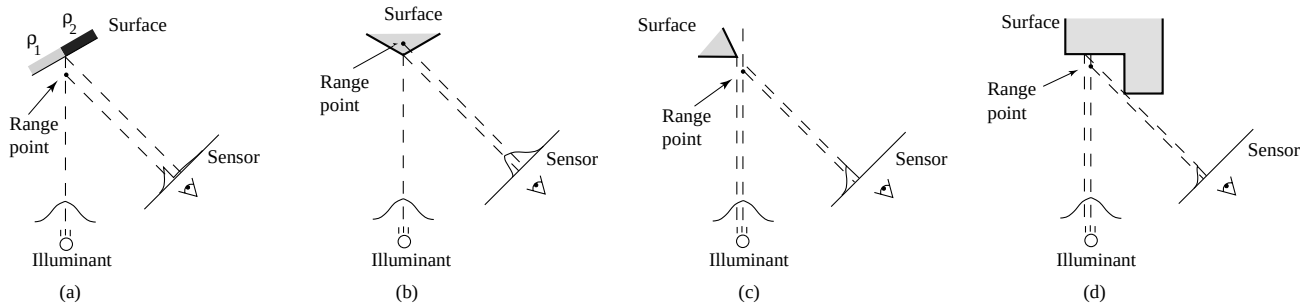


Figure 2: Range errors using traditional triangulation methods. (a) Reflectance discontinuity. (b) Corner. (c) Shape discontinuity with respect to the illumination. (d) Sensor occlusion.

as mean, median or peak of the imaged light as representative of the center. These statistics give the correct answer when the surface is perfectly planar, but they are generally inaccurate whenever the surface perturbs the shape of the illuminant.

2.1 Geometric intuition

Perturbations of the shape of the imaged illuminant occur whenever:

- The surface reflectance varies.
- The surface geometry deviates from planarity.
- The light paths to the sensor are partially occluded.
- The surface is sufficiently rough to cause laser speckle.

In Figure 2, we give examples of how the first three circumstances result in range errors even for an ideal triangulation system with infinite sensor resolution and perfect calibration. For purposes of illustration, we omit the imaging optics of Figure 1 and treat the sensor as a one dimensional orthographic sensor. Further, we assume an illuminant of Gaussian cross-section, and we use the mean for determining the center of an imaged pulse. Figure 2a shows how a step reflectance discontinuity results in range points that do not lie on the surface. Figure 2b and 2c provide two examples of shape variations resulting in range errors. Note that in Figure 2c, the center of the illuminant is not even striking a surface. In this case, a measure of the center of the pulse results in a range value, when in fact the correct answer is to return no range value whatever. Finally, Figure 2d shows the effect of occluding the line of sight between the illuminated surface and the sensor. This range error is very similar to the error encountered in Figure 2c.

The fourth source of range error is laser speckle, which arises when coherent laser illumination bounces off of a surface that is rough compared to a wavelength [7]. The surface roughness introduces random variations in optical path lengths, causing a random interference pattern throughout space and at the sensor. The result is an imaged pulse with a noise component that affects the mean pulse detection, causing range errors even from a planar target.

2.2 Quantifying the error

To quantify the errors inherent in using mean pulse analysis, we have computed the errors introduced by reflectance and shape variations for an ideal triangulation system with a single Gaussian illuminant. We take the beam width, w, to be the distance between the beam center and the e^{-2} point of the irradiance profile, a convention common to the optics literature. We present the range errors in a scale invariant form by dividing all distances by the beam width. Figure 3 illustrates the maximum deviation from planarity introduced by scanning reflectance discontinuities of varying step magnitudes for varying triangulation angles. As the size of the step increases, the error increases correspondingly. In addition, smaller triangulation angles, which are desirable for reducing the likelihood of missing data due to sensor occlusions, actually result in larger range errors. This result is not surprising, as sensor mean positions are converted to depths through a division by $sin\theta$, where θ is the triangulation angle, so that errors in mean detection translate to larger range errors for smaller triangulation angles.

Figure 4 shows the effects of a corner on range error, where the error is taken to be the shortest distance between the computed range data and the exact corner point. The corner is oriented so that the illumination direction bisects the corner's angle as shown in Figure 2b. As we might expect, a sharper corner results in greater compression of the left side of the imaged Gaussian relative to the right side, pushing the mean further to the right on the sensor and pushing the triangulated point further behind the corner. In this case, the triangulation angle has little effect as the division by $sin\theta$ is offset almost exactly by the smaller observed left/right pulse compression imbalance.

One possible strategy for reducing these errors would be to decrease the width of the beam and increase the resolution of the sensor. However, diffraction limits prevent us from focusing the beam to an arbitrary width. The limits on focusing a Gaussian beam with spherical lenses are well known [15]. In recent years, Bickel, et al, [3] have explored the use of axicons (e.g., glass cones and other surfaces of revolution) to attain tighter focus of a Gaussian beam. The refracted beam, however, has a zeroth order Bessel function cross-section; i.e., it has numerous side-lobes of non-negligible irradiance. The influence of these side-lobes is not well-documented and would seem to complicate triangulation.

3 A New Method: Spacetime Analysis

The previous section clearly demonstrates that analyzing each imaged pulse using a low order statistic leads to systematic range errors. We have found that these errors can be reduced or eliminated by analyzing the time evolution of the pulses.

3.1 Geometric intuition

Figure 5 illustrates the principle of spacetime analysis for a laser triangulation scanner with Gaussian illuminant and orthographic sen-

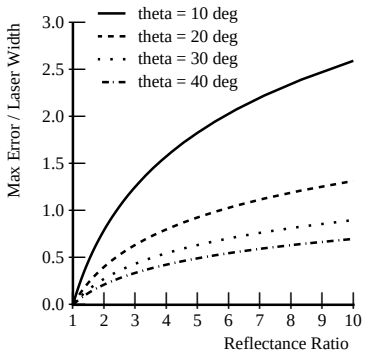


Figure 3: Plot of errors due to reflectance discontinuities for varying triangulation angles (theta).

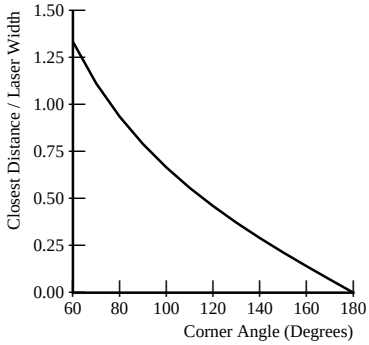


Figure 4: Plot of errors due to corners.

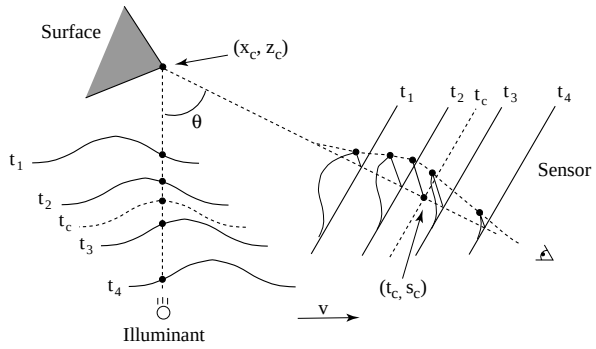


Figure 5: Spacetime mapping of a Gaussian illuminant. As the light sweeps across the corner point, the sensor images the shape of the illuminant over time.

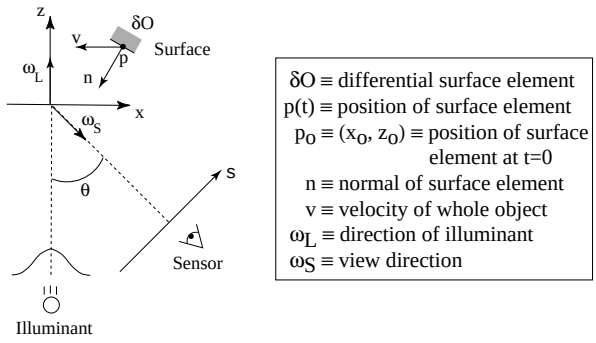


Figure 6: Triangulation scanner coordinate system. A depiction of the coordinate systems and the vectors relevant to a moving differential element.

sor as it translates across the edge of an object. As the scanner steps to the right, the sensor images a smaller and smaller portion of the laser cross-section. By time t_3, the sensor no longer images the center of the illuminant, and conventional methods of range estimation fail. However, if we look along the lines of sight from the corner to the laser and from the corner to the sensor, we see that the profile of the laser is being imaged *over time* onto the sensor (indicated by the dotted Gaussian envelope). Thus, we can find the coordinates of the corner point (x_c, z_c) by searching for the mean of a Gaussian along a constant line of sight through the sensor images. We can express the coordinates of this mean as a time and a position on the sensor, where the time is in general between sensor frames and the position is between sensor pixels. The position on the sensor indicates a depth, and the time indicates the lateral position of the center of the illuminant. In the example of Figure 5, we find that the spacetime Gaussian corresponding to the exact corner has its mean at position s_c on the sensor at a time t_c between t_2 and t_3 during the scan. We extract the corner's depth by triangulating the center of the illuminant with the line of sight corresponding to the sensor coordinate s_c, while the corner's horizontal position is proportional to the time t_c.

3.2 A complete derivation

For a more rigorous analysis, we consider the time evolution of the irradiance from a translating differential surface element, δO, as recorded at the sensor. We refer the reader to Figure 6 for a description of coordinate systems; note that in contrast to the previous section, the surface element is translating instead of the illuminant-sensor assembly. The element has a normal $\hat{n}$ and an initial position $\vec{p}_o$ and is translating with velocity $\vec{v}$, so that:

$$\vec{p}(t) = \vec{p}_o + t\vec{v} \tag{1}$$

Our objective is to compute the coordinates $\vec{p}_o = (x_o, z_o)$ given the temporal irradiance variations on the sensor. For simplicity, we assume that $\vec{v} = (-v, 0)$. The illuminant we consider is a laser with a unidirectional Gaussian radiance profile. We can describe the total radiance reflected from the element to the sensor as:

$$L(\vec{p}(t), \hat{\omega}_S) = f_r(\hat{\omega}_L, \hat{\omega}_S)|\hat{n} \cdot \hat{\omega}_L| I_L e^{\frac{-2(x_o - vt)^2}{w^2}} \tag{2}$$

where f_r is the bidirectional reflection distribution function (BRDF) of the point $\vec{p}_o$, $|\hat{n} \cdot \hat{\omega}_L|$ is the cosine of the angle between the surface and illumination. The remaining terms describe a point moving in the x-direction under the Gaussian illuminant of width w and power I_L.

Projecting the point $\vec{p}(t)$ onto the sensor, we find:

$$s = (x_o - vt)cos\theta - z_o sin\theta \tag{3}$$

where s is the position on the sensor and θ is the angle between the sensor and laser directions. We combine Equations 2-3 to give us an equation for the irradiance observed at the sensor as a function of time and position on the sensor:

$$E_S(t, s) \quad = \quad f_r(\hat{\omega}_L, \hat{\omega}_S)|\hat{n} \cdot \hat{\omega}_L| I_L e^{\frac{-2(x_o - vt)^2}{w^2}}$$

$$\delta(s - (x_o - vt)cos\theta - z_o sin\theta) \quad (4)$$

To simplify this expression, we condense the light reflection terms into one measure:

$$\beta \equiv f_r(\hat{\omega}_L, \hat{\omega}_S)|\hat{n} \cdot \hat{\omega_L}| \quad (5)$$

which we will refer to as the reflectance coefficient of point $\vec{p}$ for the given illumination and viewing directions. We also note that $x = vt$ is a measure of the relative x-displacement of the point during a scan, and $z = s/sin\theta$ is the relation between sensor coordinates and depth values along the center of the illuminant. Making these substitutions we have:

$$\begin{aligned} E_S(x,z) &= \beta I_L e^{\frac{-2(x-x_o)^2}{w^2}} \\ &\quad \delta((x - x_o)cos\theta + (z - z_o)sin\theta) \end{aligned} \quad (6)$$

This equation describes a Gaussian running along a *tilted line* through the spacetime sensor plane or "spacetime image". We define the "spacetime image" to be the image whose columns are filled with sensor scanlines that evolve over time. Through the substitutions above, position within a column of this image represents displacement in depth, and position within a row represents time or displacement in lateral position. Figure 7 shows the theoretical spacetime image of a single point based on the derivation above, while Figures 8a and 8b shows the spacetime image generated during a real scan. From Figure 7, we see that the tilt angle is $-\theta$ with respect to the z-axis, and the width of the Gaussian along the line is:

$$w' = w/cos\theta \quad (7)$$

The peak value of the Gaussian is βI_L, and its mean along the line is located at (x_o, z_o), the exact location of the range point. Note that the angle of the line and the width of the Gaussian are solely determined by the fixed parameters of the scanner, *not* the position, orientation, or BRDF of the surface element.

Thus, extraction of range points should proceed by computing low order statistics along tilted lines through the sensor spacetime image, rather than along columns (scanlines) as in the conventional method. As a result, we can determine the position of the surface element independently of the orientation and BRDF of the element and independently of any other nearby surface elements. In theory, the decoupling of range determination from local shape and reflectance is complete. In practice, optical systems and sensors have filtering and sampling properties that limit the ability to resolve neighboring points. In Figure 8d, for instance, the extracted edges extend slightly beyond their actual bounds. We attribute this artifact to filtering which blurs the exact cutoffs of the edges into neighboring pixels in the spacetime image, causing us to find additional range values.

As a side effect of the spacetime analysis, the peak of the Gaussian yields the irradiance at the sensor due to the point. Thus, we automatically obtain an intensity image precisely registered to the range image.

3.3 Generalizing the geometry

We can easily generalize the previous results to other scanner geometries under the following conditions:

- The illuminant direction is constant over the path of each range point.
- The sensor is orthographic.
- The motion is purely translational.

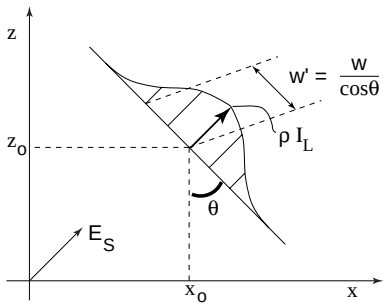


Figure 7: Spacetime image of a point passing through a Gaussian illuminant.

These conditions ensure that the reflectance coefficient, $\beta = f_r(\hat{\omega}_L, \hat{\omega}_S)|\hat{n} \cdot \hat{\omega}_L|$, is constant. Note that the illumination need only be directional; coherent or incoherent light of any pattern is acceptable. Further, the translational motion need not be of constant speed, only constant direction; we can correct for known variations in speed by applying a suitable warp to the spacetime image.

We can weaken each of these restrictions if β does not vary appreciably for each point as it passes through the illuminant. A perspective sensor is suitable if the changes in viewing directions are relatively small for neighboring points inside the illuminant. This assumption of "local orthography" has yielded excellent results in practice. In addition, we can tolerate a rotational component to the motion as long as the radius of curvature of the point path is large relative to the beam width, again minimizing the effects on β.

3.4 Correcting laser speckle

The discussion in sections 3.1-3.3 show how we can go about extracting accurate range data in the presence of shape and reflectance variations, as well as occlusions. But what about laser speckle? Empirical observation of the time evolution of the speckle pattern with our optical triangulation scanner strongly suggests that the image of laser speckle moves as the surface moves. The streaks in the spacetime image of Figure 8b correspond to speckle noise, for the object has uniform reflectance and should result in a spacetime image with uniform peak amplitudes. These streaks are tilted precisely along the direction of the spacetime analysis, indicating that the speckle noise adheres to the surface of the object and behaves as a noisy reflectance variation. Other researchers have observed a "stationary speckle" phenomenon as well [1]. Proper analysis of this problem is an open question, likely to be resolved with the study of the governing equations of scalar diffraction theory for imaging of a rough translating surface under coherent Gaussian beam illumination [6].

4 Implementation

We have implemented the spacetime analysis presented in the previous section using a commercial laser triangulation scanner and a real-time digital video recorder.

4.1 Hardware

The optical triangulation system we use is a Cyberware MS platform scanner. This scanner collects range data by casting a laser stripe on the object and by observing reflections with a CCD camera positioned at an angle of 30^o with respect to the plane of the

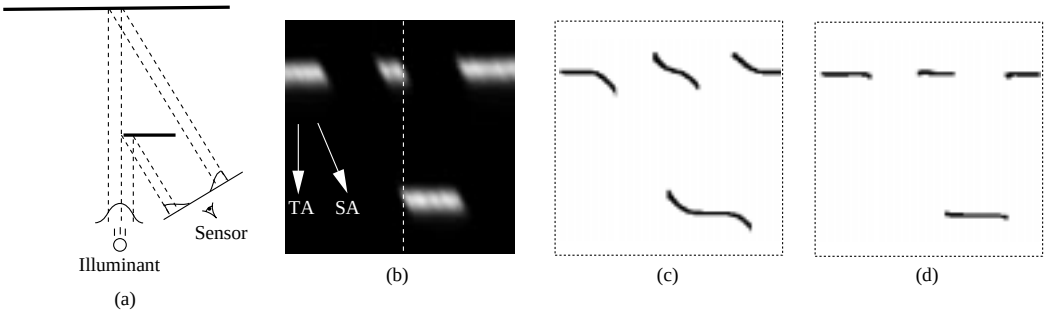


Figure 8: From geometry to spacetime image to range data. (a) The original geometry. (b) The resulting spacetime image. TA indicates the direction of traditional analysis, while SA is the direction of the spacetime analysis. The dotted line corresponds to the scanline generated at the instant shown in (a). (c) Range data after traditional mean analysis. (d) Range data after spacetime analysis.

laser. The platform can either translate or rotate an object through the field of view of the triangulation optics. The laser width varies from 0.8 mm to 1.0 mm over the field of view which is approximately 30 cm in depth and 30 cm in height. Each CCD pixel images a portion of the laser plane roughly 0.5 mm by 0.5 mm. Although the Cyberware scanner performs a form of peak detection in real time, we require the actual video frames of the camera for our analysis. We capture these frames with an Abekas A20 video digitizer and an Abekas A60 digital video disk, a system that can acquire 486 by 720 size frames at 30 Hz. These captured frames have approximately the same resolution as the Cyberware range camera, though they represent a resampling of the reconstructed CCD output.

4.2 Algorithms

Using the principles of section 3, we can devise a procedure for extracting range data from spacetime images:

1. Perform the range scan and capture the spacetime images.
2. Rotate the spacetime images by $-\theta$.
3. Find the statistics of the Gaussians in the rotated coordinates.
4. Rotate the means back to the original coordinates.

In order to implement step 1 of this algorithm, we require a sequence of CCD images. Most commercial optical triangulation systems discard each CCD image after using it (e.g. to compute a stripe of the range map). As described in section 4.1, we have assembled the necessary hardware to record the CCD frames. In section 3, we discussed a one dimensional sensor scenario and indicated that perspective imaging could be treated as locally orthographic. For a two dimensional sensor, we can imagine the horizontal scanlines as separate one dimensional sensors with varying vertical (y) offsets. Each scanline generates a spacetime image, and by stacking the spacetime images one atop another, we define a spacetime *volume*. In general, we must perform our analysis along the paths of points, paths which may cross scanlines within the spacetime volume. However, we have observed for our system that the illuminant is sufficiently narrow and the perspective of the range camera sufficiently weak, that these paths essentially remain within scanlines. This observation allows us to perform our analysis on each spacetime image separately.

In step 2, we rotate the spacetime images so that Gaussians are vertically aligned. In a practical system with different sampling rates in x and z, the correct rotation angle can be computed as:

$$tan\theta = \frac{\tau_z}{\tau_x} tan\theta_T \tag{8}$$

where θ is the new rotation angle, τ_x and τ_z are the sample spacing in x and z respectively, and θ_T is the triangulation angle. In order to determine the rotation angle, θ, for a given scanning rate and region of the field of view of our Cyberware scanner, we first determined the local triangulation angle and the sample spacings in depth (z) and lateral position (x). Equation 8 then yields the desired angle.

In step 3, we compute the statistics of the Gaussians along each rotated spacetime image raster. Our method of choice for computing these statistics is a least squares fit of a parabola to the log of the data. We have experimented with fitting the data directly to Gaussians using the Levenberg-Marquardt non-linear least squares algorithm [13], but the results have been substantially the same as the log-parabola fits. The Gaussian statistics consist of a mean, which corresponds to a range point, as well as a width and a peak amplitude, both of which indicate the reliability of the data. Widths that are far from the expected width and peak amplitudes near the noise floor of the sensor imply unreliable data which may be downweighted or discarded during later processing (e.g., when combining multiple range meshes [18]). For the purposes of this paper, we discard unreliable data.

Finally, in step 4, we rotate the range points back into the global coordinate system.

Traditionally, researchers have extracted range data at sampling rates corresponding to one range point per sensor scanline per unit time. Interpolation of shape between range points has consisted of fitting primitives (e.g., linear interpolants like triangles) to the range points. Instead, we can regard the spacetime volume as the primary source of information we have about an object. After performing a real scan, we have a sampled representation of the spacetime volume, which we can then reconstruct to generate a continuous function. This function then acts as our range oracle, which we can query for range data at a sampling rate of our choosing. In practice, we can magnify the sampled spacetime volume prior to applying the range imaging steps described above. The result is a range grid with a higher sampling density based directly on the imaged light reflections.

5 Results

5.1 Reflectance correction

To evaluate the tolerance of the spacetime method to changes in reflectance, we performed two experiments, one quantitative and the other qualitative. For the first experiment, we generated planar cards with step reflectance changes varying from about 1:1 to 10:1 and scanned them at an angle of 30^o (roughly facing the sen-

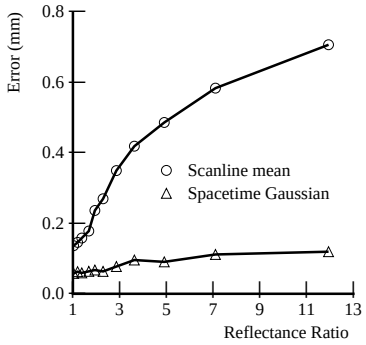


Figure 9: Measured error due to varying reflectance steps.

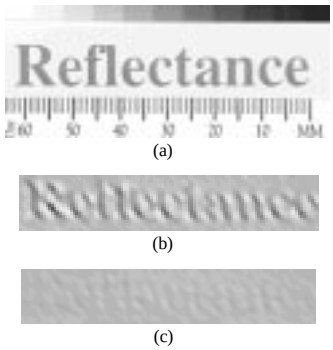


Figure 10: Reflectance card. (a) Photograph of a planar card with the word "Reflectance" printed on it, and shaded renderings of the range data generated by (b) mean pulse analysis and (c) spacetime analysis.

sor). Figure 9 shows a plot of maximum deviations from planarity when using traditional per scanline mean analysis and our spacetime analysis. The spacetime method has clearly improved over the old method, yielding up to 85% reductions in range errors.

For qualitative comparison, we produced a planar sheet with the word "Reflectance" printed on it. Figure 10 shows the results. The old method yields a surface with the characters well-embossed into the geometry, whereas the spacetime method yields a much more planar surface indicating successful decoupling of geometry and reflectance.

5.2 Shape correction

We conducted several experiments to evaluate the effects of shape variation on range acquisition. In the first experiment, we generated corners of varying angles by abutting sharp edges of machined aluminum wedges which are painted white. Figure 11 shows the range errors that result for traditional and spacetime methods. Again, we see an increase in accuracy, though not as great as in the reflectance case.

We also scanned two 4 mm strips of paper at an angle of 30^o (roughly facing the sensor) to examine the effects of depth continuity. Figure 12b shows the "edge curl" observed with the old method, while the spacetime method in Figure 12c shows a significant reduction of this artifact under spacetime analysis. We have found that the spacetime method reduces the length of the edge curl from an average of 1.1 mm to an average of approximately 0.35

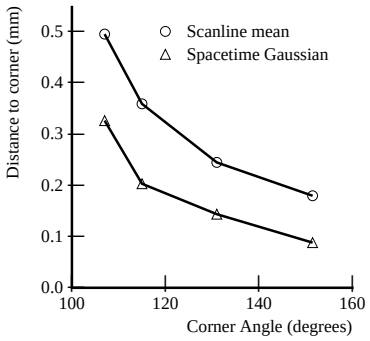


Figure 11: Measured error due to corners of varying angles.

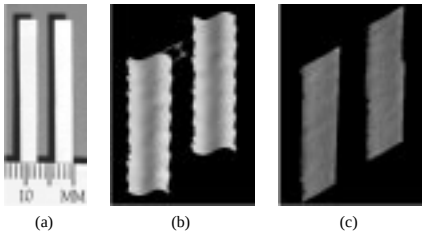


Figure 12: Depth discontinuities and edge curl. (a) Photograph of two strips of paper, and shaded renderings of the range data generated by (b) mean pulse analysis and (c) spacetime analysis. The "edge curl" indicated by the hash-marks in (b) is 1.1mm.

mm.

Finally, we impressed the word "shape" onto a plastic ribbon using a commonly available label maker. In Figure 10, we wanted the word "Reflectance" to disappear because it represented changes in reflectance rather than in geometry. In Figure 13, we want the word "Shape" to stay because it represents real geometry. Furthermore, we wish to resolve it as highly as possible. Figure 13 shows the result. Using the scanline mean method, the word is barely visible. Using the new spacetime analysis, the word becomes legible.

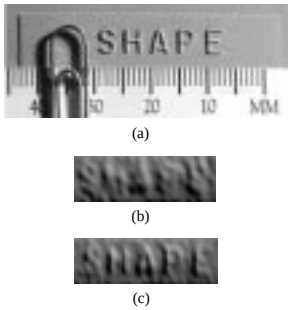


Figure 13: Shape ribbon. (a) Photograph of a surface with raised lettering (letters are approx. 0.3 mm high), and renderings of the range data generated by (b) mean pulse analysis and (c) spacetime analysis.

5.3 Speckle reduction

We performed range scans on the planar surfaces and generated range points using the traditional and spacetime methods. After fitting planes to range points, we found a 30-60% reduction in average deviation from planarity when using the spacetime analysis.

5.4 A complex object

Figure 14 shows the results of scanning a model tractor. Figure 14b is a rendering of the data generated by the Cyberware scanner hardware and is particularly noisy. This added noisiness results from the method of pulse analysis performed by the hardware, a method similar to peak detection. Peak detection is especially susceptible to speckle noise, because it extracts a range point based on a single value or small neighborhood of values on a noisy curve. Mean analysis tends to average out the speckle noise, resulting in smoother range data as shown in Figure 14c. Figure 14d shows our spacetime results and Figure 14e shows the spacetime results with 3X interpolation and resampling of the spacetime volume as described in section 4.2. Note the sharper definition of features on the body of the tractor and less jagged edges in regions of depth discontinuity.

5.5 Remaining sources of error

The results we presented in this section clearly show that the spacetime analysis yields more accurate range data, but the results are imperfect due to system limitations. These limitations include:

- CCD noise
- Finite sensor resolution
- Optical blurring and electronic filtering
- Quantization errors
- Calibration errors
- Surface-surface inter-reflections

In addition, we observed some electronic artifacts in our Cyberware scanner that influenced our results. We expect, however, that any measures taken to reduce the effects of the limiting factors described above will lead to higher accuracy. By contrast, if one uses traditional methods of range extraction, then increasing sensor resolution and reducing the effects of filtering alone will *not* significantly increase tolerance to reflectance and shape changes when applying the traditional methods of range extraction.

6 Conclusion

We have described several of the systematic limitations in traditional methods of range acquisition with optical triangulation range scanners, including intolerance to reflectance and shape changes and speckle noise. By analyzing the time evolution of the reflected light imaged onto the sensor, we have shown that distortions induced by shape and reflectance changes can be corrected, while the influence of laser speckle can be reduced. In practice, we have demonstrated that we can significantly reduce range distortions with existing hardware. Although the spacetime method does not completely eliminate range artifacts in practice, it has proven to reduce the artifacts in all experiments we have conducted.

In future work, we plan to incorporate the improved range data with algorithms that integrate partial triangulation scans into complete, unified meshes. We expect this improved data to ease the process of estimating topology, especially in areas of high curvature which are prone to edge curl artifacts. We will also investigate methods for increasing the resolution of the existing hardware by registering and deblurring multiple spacetime images [9]. Finally, we hope to apply the results of scalar diffraction theory to put the achievement of speckle reduction on sound theoretical footing.

Acknowledgments

We thank the people of Cyberware for the use of the range scanner and for their help in accessing the raw video output from the range camera.

References

[1] R. Baribeau and M. Rioux. Influence of speckle on laser range finders. *Applied Optics*, 30(20):2873–2878, July 1991.

[2] Paul Besl. *Advances in Machine Vision*, chapter 1 - Active optical range imaging sensors, pages 1–63. Springer-Verlag, 1989.

[3] G. Bickel, G. Haulser, and M. Maul. Triangulation with expanded range of depth. *Optical Engineering*, 24(6):975–977, December 1985.

[4] M. Buzinski, A. Levine, and W.H. Stevenson. Performance characteristics of range sensors utilizing optical triangulation. In *Proceedings of the IEEE 1992 National Aerospace and Electronics Conference, NAECON 1992*, pages 1230–1236, May 1992.

[5] R.G. Dorsch, G. Hausler, and J.M. Herrmann. Laser triangulation: fundamental uncertainty in distance measurement. *Applied Optics*, 33(7):1306–1314, March 1994.

[6] Joseph W. Goodman. *Introduction to Fourier optics*. McGraw-Hill, 1968.

[7] J.W. Goodman. *Laser Speckle and Related Phenomena*, chapter 1 - Statistical properties of laser speckle patterns, pages 9–76. Springer-Verlag, 1984.

[8] G. Hausler and W. Heckel. Light sectioning with large depth and high resolution. *Applied Optics*, 27(24):5165–5169, Dec 1988.

[9] M. Irani and S. Peleg. Improving resolution by image registration. *CVGIP: Graphical Models and Image Processing*, 53(3):231–239, May 1991.

[10] R.A. Jarvis. A perspective on range-finding techniques for computer vision. *IEEE Trans. Pattern Analysis Mach. Intell.*, 5:122–139, March 1983.

[11] T Kanade, A Gruss, and L Carley. A very fast vlsi rangefinder. In *1991 IEEE International Conference on Robotics and Automation*, volume 39, pages 1322–1329, April 1991.

[12] J.L. Mundy and G.B. Porter. *Three-dimensional machine vision*, chapter 1 - A three-dimensional sensor based on structured light, pages 3–61. Kluver Academic Publishers, 1987.

[13] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1986.

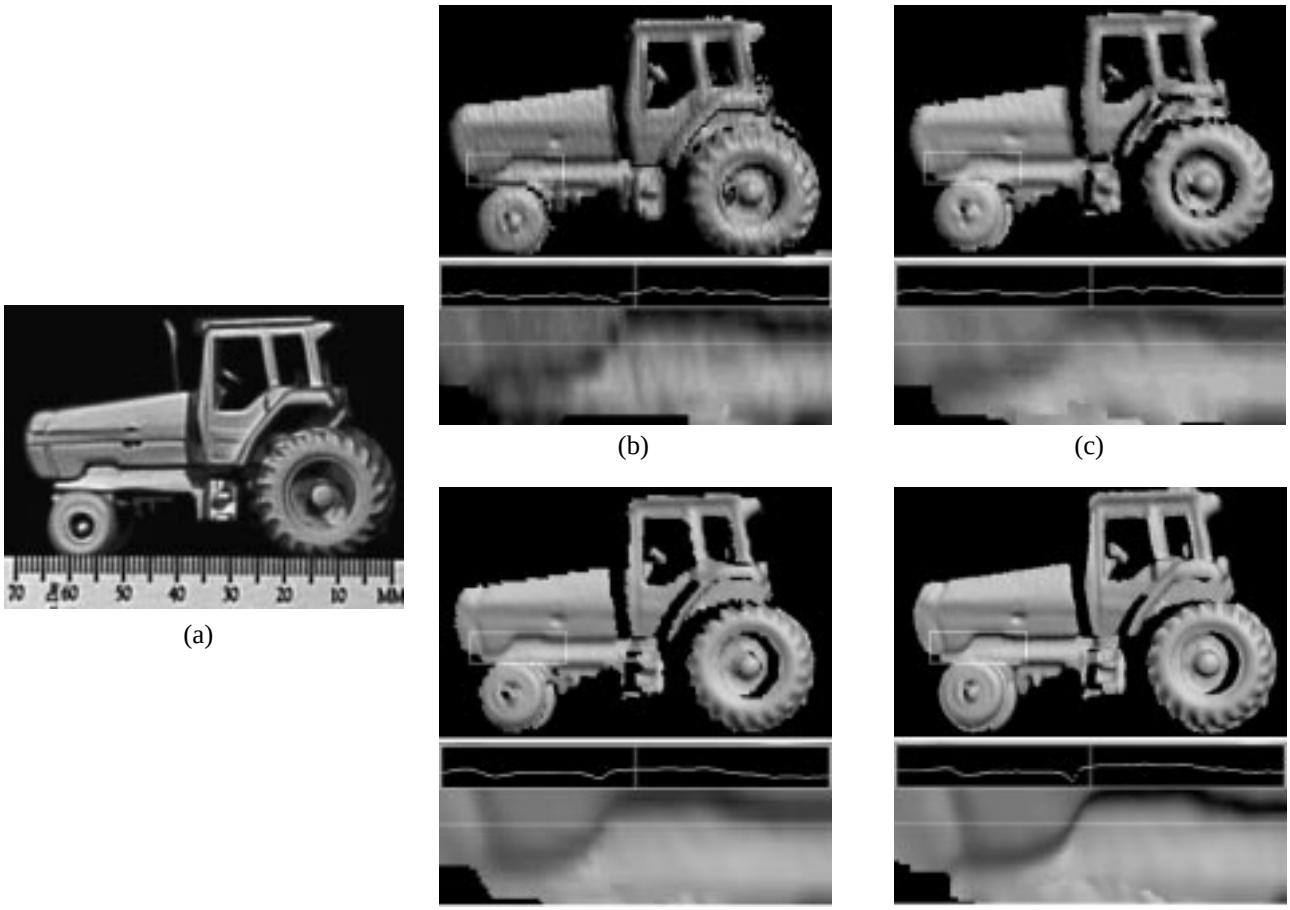


Figure 14: Model tractor. (a) Photograph of original model and shaded renderings of range data generated by (b) the Cyberware scanner hardware, (c) mean pulse analysis, (d) our spacetime analysis, and (e) the spacetime analysis with 3X interpolation of the spacetime volume before fitting the Gaussians. Below each of the renderings is a blow-up of one section of the tractor body (indicated by rectangle on rendering) with a plot of one row of pixel intensities.

[14] M. Rioux, G. Bechthold, D. Taylor, and M. Duggan. Design of a large depth of view three-dimensional camera for robot vision. *Optical Engineering*, 26(12):1245–1250, Dec 1987.

[15] A.E. Siegman. *Lasers*. University Science Books, 1986.

[16] M. Soucy, D. Laurendeau, D. Poussart, and F. Auclair. Behaviour of the center of gravity of a reflected gaussian laser spot near a surface reflectance discontinuity. *Industrial Metrology*, 1(3):261–274, Sept 1990.

[17] T. Strand. Optical three dimensional sensing. *Optical Engineering*, 24(1):33–40, Jan-Feb 1983.

[18] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *SIGGRAPH 94 Conference Proceedings*, pages 311–318, July 1994.

A Volumetric Method for Building Complex Models from Range Images

Brian Curless and Marc Levoy
Stanford University

Abstract

A number of techniques have been developed for reconstructing surfaces by integrating groups of aligned range images. A desirable set of properties for such algorithms includes: incremental updating, representation of directional uncertainty, the ability to fill gaps in the reconstruction, and robustness in the presence of outliers. Prior algorithms possess subsets of these properties. In this paper, we present a volumetric method for integrating range images that possesses all of these properties.

Our volumetric representation consists of a cumulative weighted signed distance function. Working with one range image at a time, we first scan-convert it to a distance function, then combine this with the data already acquired using a simple additive scheme. To achieve space efficiency, we employ a run-length encoding of the volume. To achieve time efficiency, we resample the range image to align with the voxel grid and traverse the range and voxel scanlines synchronously. We generate the final manifold by extracting an isosurface from the volumetric grid. We show that under certain assumptions, this isosurface is optimal in the least squares sense. To fill gaps in the model, we tessellate over the boundaries between regions seen to be empty and regions never observed.

Using this method, we are able to integrate a large number of range images (as many as 70) yielding seamless, high-detail models of up to 2.6 million triangles.

CR Categories: I.3.5 [Computer Graphics] Computational Geometry and Object Modeling
Additional keywords: Surface fitting, three-dimensional shape recovery, range image integration, isosurface extraction

1 Introduction

Recent years have witnessed a rise in the availability of fast, accurate range scanners. These range scanners have provided data for applications such as medicine, reverse engineering, and digital film-making. Many of these devices generate *range images*; i.e., they produce depth values on a regular sampling lattice. Figure 1 illustrates how an optical triangulation scanner can be used to acquire a range image. By connecting nearest neighbors with triangular elements, one can construct a *range surface* as shown in Figure 1d. Range images are typically formed by sweeping a 1D or 2D sensor linearly across an object or circularly around it, and generally do not contain enough information to reconstruct the entire object being scanned. Accordingly, we require algorithms that can merge multiple range images into a single description of the surface. A set of desirable properties for such a surface reconstruction algorithm includes:

- *Representation of range uncertainty*. The data in range images typically have asymmetric error distributions with primary directions along sensor lines of sight, as illustrated for optical triangulation in Figure 1a. The method of range integration should reflect this fact.
- *Utilization of all range data*, including redundant observations of each object surface. If properly used, this redundancy can reduce sensor noise.
- *Incremental and order independent updating*. Incremental updates allow us to obtain a reconstruction after each scan or small set of scans and allow us to choose the next best orientation for scanning. Order independence is desirable to ensure that results are not biased by earlier scans. Together, they allow for straightforward parallelization.
- *Time and space efficiency*. Complex objects may require many range images in order to build a detailed model. The range images and the model must be represented efficiently and processed quickly to make the algorithm practical.
- *Robustness*. Outliers and systematic range distortions can create challenging situations for reconstruction algorithms. A robust algorithm needs to handle these situations without catastrophic failures such as holes in surfaces and self-intersecting surfaces.
- *No restrictions on topological type*. The algorithm should not assume that the object is of a particular genus. Simplifying assumptions such as "the object is homeomorphic to a sphere" yield useful results in only a restricted class of problems.
- *Ability to fill holes in the reconstruction*. Given a set of range images that do not completely cover the object, the surface reconstruction will necessarily be incomplete. For some objects, no amount of scanning would completely cover the object, because some surfaces may be inaccessible to the sensor. In these cases, we desire an algorithm that can automatically fill these holes with plausible surfaces, yielding a model that is both "watertight" and esthetically pleasing.

In this paper, we present a volumetric method for integrating range images that possesses all of these properties. In the next section, we review some previous work in the area of surface reconstruction. In section 3, we describe the core of our volumetric algorithm. In section 4, we show how this algorithm can be used to fill gaps in the reconstruction using knowledge about the emptiness of space. Next, in section 5, we describe how we implemented our volumetric approach so as to keep time and space costs reasonable. In section 6, we show the results of surface reconstruction from many range images of complex objects. Finally, in section 7 we conclude and discuss limitations and future directions.

2 Previous work

Surface reconstruction from dense range data has been an active area of research for several decades. The strategies have proceeded along two basic directions: reconstruction from unorganized points, and

Authors' Address: Computer Science Department, Stanford University, Stanford, CA 94305
E-mail: {curless,levoy}@cs.stanford.edu
World Wide Web: http://www-graphics.stanford.edu

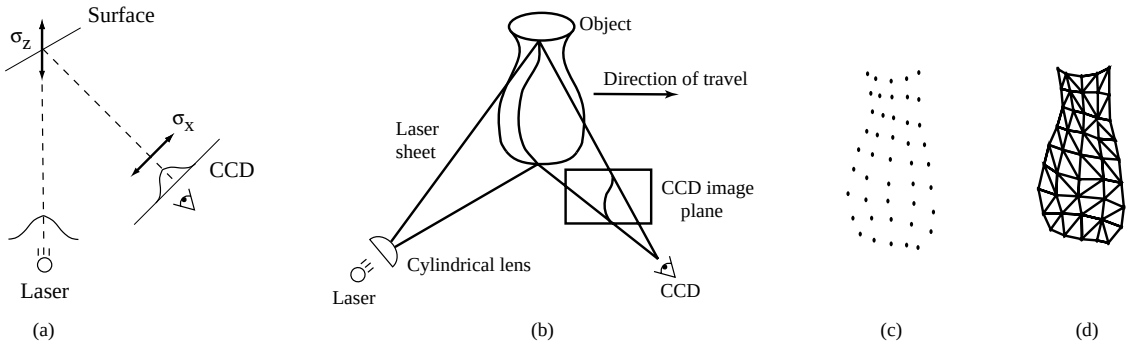


Figure 1. From optical triangulation to a range surface. (a) In 2D, a narrow laser beam illuminates a surface, and a linear sensor images the reflection from an object. The center of the image pulse maps to the center of the laser, yielding a range value. The uncertainty, σ_x, in determining the center of the pulse results in range uncertainty, σ_z along the laser's line of sight. When using the spacetime analysis for optical triangulation [6], the uncertainties run along the lines of sight of the CCD. (b) In 3D, a laser stripe triangulation scanner first spreads the laser beam into a sheet of light with a cylindrical lens. The CCD observes the reflected stripe from which a depth profile is computed. The object sweeps through the field of view, yielding a range image. Other scanner configurations rotate the object to obtain a cylindrical scan or sweep a laser beam or stripe over a stationary object. (c) A range image obtained from the scanner in (b) is a collection of points with regular spacing. (d) By connecting nearest neighbors with triangles, we create a piecewise linear range surface.

reconstruction that exploits the underlying structure of the acquired data. These two strategies can be further subdivided according to whether they operate by reconstructing parametric surfaces or by reconstructing an implicit function.

A major advantage of the unorganized points algorithms is the fact that they do not make any prior assumptions about connectivity of points. In the absence of range images or contours to provide connectivity cues, these algorithms are the only recourse. Among the parametric surface approaches, Boissanat [2] describes a method for Delaunay triangulation of a set of points in 3-space. Edelsbrunner and Mücke [9] generalize the notion of a convex hull to create surfaces called alpha-shapes. Examples of implicit surface reconstruction include the method of Hoppe, et al [16] for generating a signed distance function followed by an isosurface extraction. More recently, Bajaj, et al [1] used alpha-shapes to construct a signed distance function to which they fit implicit polynomials. Although unorganized points algorithms are widely applicable, they discard useful information such as surface normal and reliability estimates. As a result, these algorithms are well-behaved in smooth regions of surfaces, but they are not always robust in regions of high curvature and in the presence of systematic range distortions and outliers.

Among the structured data algorithms, several parametric approaches have been proposed, most of them operating on range images in a polygonal domain. Soucy and Laurendeau [25] describe a method using Venn diagrams to identify overlapping data regions, followed by re-parameterization and merging of regions. Turk and Levoy [30] devised an incremental algorithm that updates a reconstruction by eroding redundant geometry, followed by zippering along the remaining boundaries, and finally a consensus step that reintroduces the original geometry to establish final vertex positions. Rutishauser, et al [24] use errors along the sensor's lines of sight to establish consensus surface positions followed by a re-tessellation that incorporates redundant data. These algorithms typically perform better than unorganized point algorithms, but they can still fail catastrophically in areas of high curvature, as exemplified in Figure 9.

Several algorithms have been proposed for integrating structured data to generate implicit functions. These algorithms can be classified as to whether voxels are assigned one of two (or three) states or are samples of a continuous function. Among the discrete-state volumetric algorithms, Connolly [4] casts rays from a range image accessed as a quad-tree into a voxel grid stored as an octree, and generates results for synthetic data. Chien, et al [3] efficiently generate octree models under the severe assumption that all views are taken from the directions corresponding to the 6 faces of a cube. Li and Crebbin [19] and Tarbox and Gottschlich [28] also describe methods for generating binary voxel grids from range images. None of these methods has been used to generate surfaces. Further, without an underlying continuous function, there are no mechanism for representing range uncertainty or for combining overlapping, noisy range surfaces.

The last category of our taxonomy consists of implicit function methods that use samples of a continuous function to combine structured data. Our method falls into this category. Previous efforts in this area include the work of Grosso, et al [12], who generate depth maps from stereo and average them into a volume with occupancy ramps of varying slopes corresponding to uncertainty measures; they do not, however, perform a final surface extraction. Succi, et al [26] create depth maps from stereo and optical flow and integrate them volumetrically using a straight average. The details of his method are unclear, but they appear to extract an isosurface at an arbitrary threshold. In both the Grosso and Succi papers, the range maps are sparse, the directions of range uncertainty are not characterized, they use no time or space optimizations, and the final models are of low resolution. Recently, Hilton, et al [14] have developed a method similar to ours in that it uses weighted signed distance functions for merging range images, but it does not address directions of sensor uncertainty, incremental updating, space efficiency, and characterization of the whole space for potential hole filling, all of which we believe are crucial for the success of this approach.

Other relevant work includes the method of probabilistic occupancy grids developed by Elfes and Matthies [10]. Their volumetric space is a scalar probability field which they update using a Bayesian formulation. The results have been used for robot navigation, but not for surface extraction. A difficulty with this technique is the fact that the best description of the surface lies at the peak or ridge of the probability function, and the problem of ridge-finding is not one with robust solutions [8]. This is one of our primary motivations for taking an isosurface approach in the next section: it leverages off of well-behaved surface extraction algorithms.

The discrete-state implicit function algorithms described above also have much in common with the methods of extracting volumes from silhouettes [15] [21] [23] [27]. The idea of using backdrops to help carve out the emptiness of space is one we demonstrate in section 4.

3 Volumetric integration

Our algorithm employs a continuous implicit function, $D(\mathbf{x})$, represented by samples. The function we represent is the weighted signed distance of each point $\mathbf{x}$ to the nearest range surface along the line of

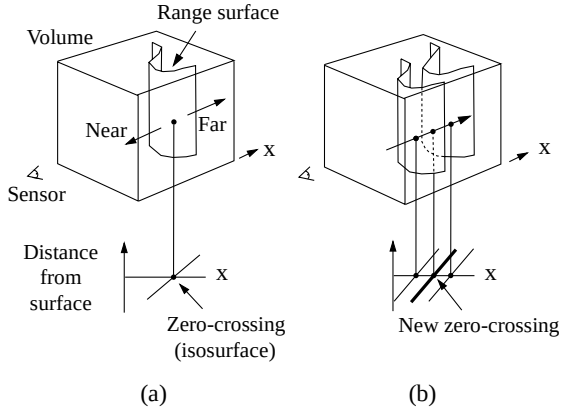


Figure 2. Unweighted signed distance functions in 3D. (a) A range sensor looking down the x-axis observes a range image, shown here as a reconstructed range surface. Following one line of sight down the x-axis, we can generate a signed distance function as shown. The zero crossing of this function is a point on the range surface. (b) The range sensor repeats the measurement, but noise in the range sensing process results in a slightly different range surface. In general, the second surface would interpenetrate the first, but we have shown it as an offset from the first surface for purposes of illustration. Following the same line of sight as before, we obtain another signed distance function. By summing these functions, we arrive at a cumulative function with a new zero crossing positioned midway between the original range measurements.

sight to the sensor. We construct this function by combining signed distance functions $d_1(\mathbf{x})$, $d_2(\mathbf{x})$, ... $d_n(\mathbf{x})$ and weight functions $w_1(\mathbf{x})$, $w_2(\mathbf{x})$, ... $w_n(\mathbf{x})$ obtained from range images 1 ... n. Our combining rules give us for each voxel a cumulative signed distance function, $D(\mathbf{x})$, and a cumulative weight $W(\mathbf{x})$. We represent these functions on a discrete voxel grid and extract an isosurface corresponding to $D(\mathbf{x}) = 0$. Under a certain set of assumptions, this isosurface is optimal in the least squares sense. A full proof of this optimality is beyond the scope of this paper, but a sketch appears in appendix A.

Figure 2 illustrates the principle of combining unweighted signed distances for the simple case of two range surfaces sampled from the same direction. Note that the resulting isosurface would be the surface created by averaging the two range surfaces along the sensor's lines of sight. In general, however, weights are necessary to represent variations in certainty across the range surfaces. The choice of weights should be specific to the range scanning technology. For optical triangulation scanners, for example, Soucy [25] and Turk [30] make the weight depend on the dot product between each vertex normal and the viewing direction, reflecting greater uncertainty when the illumination is at grazing angles to the surface. Turk also argues that the range data at the boundaries of the mesh typically have greater uncertainty, requiring more down-weighting. We adopt these same weighting schemes for our optical triangulation range data.

Figure 3 illustrates the construction and usage of the signed distance and weight functions in 1D. In Figure 3a, the sensor is positioned at the origin looking down the +x axis and has taken two measurements, r_1 and r_2. The signed distance profiles, $d_1(x)$ and $d_2(x)$ may extend indefinitely in either direction, but the weight functions, $w_1(x)$ and $w_2(x)$, taper off behind the range points for reasons discussed below.

Figure 3b is the weighted combination of the two profiles. The combination rules are straightforward:

$$D(\mathbf{x}) = \frac{\Sigma w_i(\mathbf{x}) d_i(\mathbf{x})}{\Sigma w_i(\mathbf{x})} \tag{1}$$

$$W(\mathbf{x}) = \Sigma w_i(\mathbf{x}) \tag{2}$$

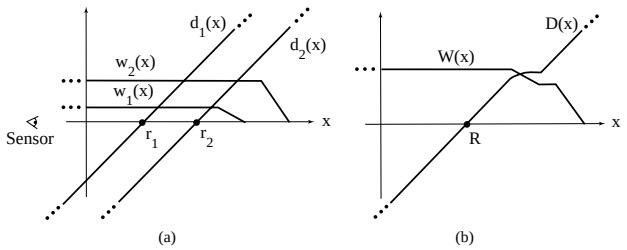


Figure 3. Signed distance and weight functions in one dimension. (a) The sensor looks down the x-axis and takes two measurements, r_1 and r_2. $d_1(x)$ and $d_2(x)$ are the signed distance profiles, and $w_1(x)$ and $w_2(x)$ are the weight functions. In 1D, we might expect two sensor measurements to have the same weight magnitudes, but we have shown them to be of different magnitude here to illustrate how the profiles combine in the general case. (b) $D(x)$ is a weighted combination of $d_1(x)$ and $d_2(x)$, and $W(x)$ is the sum of the weight functions. Given this formulation, the zero-crossing, R, becomes the weighted combination of r_1 and r_2 and represents our best guess of the location of the surface. In practice, we truncate the distance ramps and weights to the vicinity of the range points.

where, $d_i(\mathbf{x})$ and $w_i(\mathbf{x})$ are the signed distance and weight functions from the ith range image.

Expressed as an incremental calculation, the rules are:

$$D_{i+1}(\mathbf{x}) = \frac{W_i(\mathbf{x}) D_i(\mathbf{x}) + w_{i+1}(\mathbf{x}) d_{i+1}(\mathbf{x})}{W_i(\mathbf{x}) + w_{i+1}(\mathbf{x})} \tag{3}$$

$$W_{i+1}(\mathbf{x}) = W_i(\mathbf{x}) + w_{i+1}(\mathbf{x}) \tag{4}$$

where $D_i(\mathbf{x})$ and $W_i(\mathbf{x})$ are the cumulative signed distance and weight functions after integrating the ith range image.

In the special case of one dimension, the zero-crossing of the cumulative function is at a range, R given by:

$$R = \frac{\Sigma w_i r_i}{\Sigma w_i} \tag{5}$$

i.e., a weighted combination of the acquired range values, which is what one would expect for a least squares minimization.

In principle, the distance and weighting functions should extend indefinitely in either direction. However, to prevent surfaces on opposite sides of the object from interfering with each other, we force the weighting function to taper off behind the surface. There is a trade-off involved in choosing where the weight function tapers off. It should persist far enough behind the surface to ensure that all distance ramps will contribute in the vicinity of the final zero crossing, but, it should also be as narrow as possible to avoid influencing surfaces on the other side. To meet these requirements, we force the weights to fall off at a distance equal to half the maximum uncertainty interval of the range measurements. Similarly, the signed distance and weight functions need not extend far in front of the surface. Restricting the functions to the vicinity of the surface yields a more compact representation and reduces the computational expense of updating the volume.

In two and three dimensions, the range measurements correspond to curves or surfaces with weight functions, and the signed distance ramps have directions that are consistent with the primary directions of sensor uncertainty. The uncertainties that apply to range image integration include errors in alignment between meshes as well as errors inherent in the scanning technology. A number of algorithms for aligning sets of range images have been explored and shown to yield excellent results [11][30]. The remaining error lies in the scanner itself. For optical triangulation scanners, for example, this error has been shown to be ellipsoidal about the range points, with the major axis of the ellipse aligned with the lines of sight of the laser [13][24].

Figure 4 illustrates the two-dimensional case for a range curve derived from a single scan containing a row of range samples. In practice, we use a fixed point representation for the signed distance func-

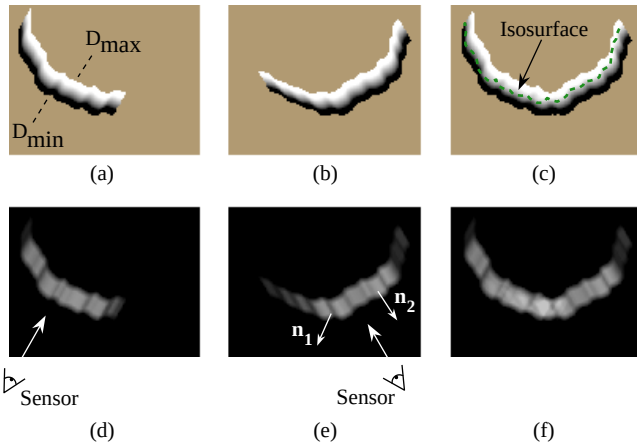


Figure 4. Combination of signed distance and weight functions in two dimensions. (a) and (d) are the signed distance and weight functions, respectively, generated for a range image viewed from the sensor line of sight shown in (d). The signed distance functions are chosen to vary between D_{min} and D_{max}, as shown in (a). The weighting falls off with increasing obliquity to the sensor and at the edges of the meshes as indicated by the darker regions in (e). The normals, $\mathbf{n_1}$ and $\mathbf{n_2}$ shown in (e), are oriented at a grazing angle and facing the sensor, respectively. Note how the weighting is lower (darker) for the grazing normal. (b) and (e) are the signed distance and weight functions for a range image of the same object taken at a 60 degree rotation. (c) is the signed distance function $D(\mathbf{x})$ corresponding to the per voxel weighted combination of (a) and (b) constructed using equations 3 and 4. (f) is the sum of the weights at each voxel, $W(\mathbf{x})$. The dotted green curve in (c) is the isosurface that represents our current estimate of the shape of the object.

tion, which bounds the values to lie between D_{min} and D_{max} as shown in the figure. The values of D_{min} and D_{max} must be negative and positive, respectively, as they are on opposite sides of a signed distance zero-crossing.

For three dimensions, we can summarize the whole algorithm as follows. First, we set all voxel weights to zero, so that new data will overwrite the initial grid values. Next, we tessellate each range image by constructing triangles from nearest neighbors on the sampled lattice. We avoid tessellating over step discontinuities (cliffs in the range map) by discarding triangles with edge lengths that exceed a threshold. We must also compute a weight at each vertex as described above.

Once a range image has been converted to a triangle mesh with a weight at each vertex, we can update the voxel grid. The signed distance contribution is computed by casting a ray from the sensor through each voxel near the range surface and then intersecting it with the triangle mesh, as shown in figure 5. The weight is computed by linearly interpolating the weights stored at the intersection triangle's vertices. Having determined the signed distance and weight we can apply the update formulae described in equations 3 and 4.

At any point during the merging of the range images, we can extract the zero-crossing isosurface from the volumetric grid. We restrict this extraction procedure to skip samples with zero weight, generating triangles only in the regions of observed data. We will relax this restriction in the next section.

4 Hole filling

The algorithm described in the previous section is designed to reconstruct the observed portions of the surface. Unseen portions of the surface will appear as holes in the reconstruction. While this result is an accurate representation of the known surface, the holes are esthetically unsatisfying and can present a stumbling block to follow-on algorithms that expect continuous meshes. In [17], for example, the authors describe a method for parameterizing patches that entails

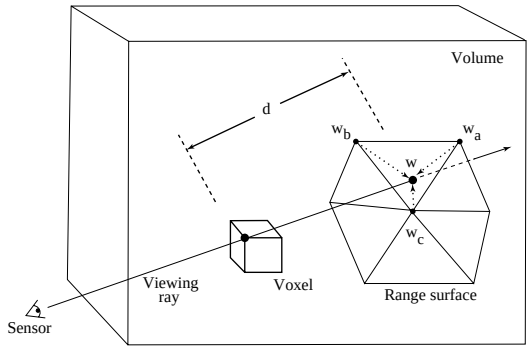


Figure 5. Sampling the range surface to update the volume. We compute the weight, w, and signed distance, d, needed to update the voxel by casting a ray from the sensor, through the voxel onto the range surface. We obtain the weight, w, by linearly interpolating the weights (w_a, w_b, and w_c) stored at neighboring range vertices. Note that for a translating sensor (like our Cyberware scanner), the sensor point is different for each column of range points.

generating evenly spaced grid lines by walking across the edges of a mesh. Gaps in the mesh prevent the algorithm from creating a fair parameterization. As another example, rapid prototyping technologies such as stereolithography typically require a "watertight" model in order to construct a solid replica [7].

One option for filling holes is to operate on the reconstructed mesh. If the regions of the mesh near each hole are very nearly planar, then this approach works well. However, holes in the meshes can be (and frequently are) highly non-planar and may even require connections between unconnected components. Instead, we offer a hole filling approach that operates on our volume, which contains more information than the reconstructed mesh.

The key to our algorithm lies in classifying all points in the volume as being in one of three states: unseen, empty, or near the surface. Holes in the surface are indicated by frontiers between unseen regions and empty regions (see Figure 6). Surfaces placed at these frontiers offer a plausible way to plug these holes (dotted in Figure 6). Obtaining this classification and generating these hole fillers leads to a straightforward extension of the algorithm described in the previous section:

1. Initialize the voxel space to the "unseen" state.
2. Update the voxels near the surface as described in the previous section. As before, these voxels take on continuous signed distance and weight values.
3. Follow the lines of sight back from the observed surface and mark the corresponding voxels as "empty". We refer to this step as *space carving*.
4. Perform an isosurface extraction at the zero-crossing of the signed distance function. Additionally, extract a surface between regions seen to be empty and regions that remain unseen.

In practice, we represent the unseen and empty states using the function and weight fields stored on the voxel lattice. We represent the unseen state with the function values $D(\mathbf{x}) = D_{max}, W(\mathbf{x}) = 0$ and the empty state with the function values $D(\mathbf{x}) = D_{min}, W(\mathbf{x}) = 0$, as shown in Figure 6b. The key advantage of this representation is that we can use the same isosurface extraction algorithm we used in the previous section without the restriction on interpolating voxels of zero weight. This extraction finds both the signed distance and hole fill isosurfaces and connects them naturally where they meet, i.e., at the corners in Figure 6a where the dotted red line meets the dashed green line. Note that the triangles that arise from interpolations across voxels of zero weight are distinct from the others: they are hole fillers.

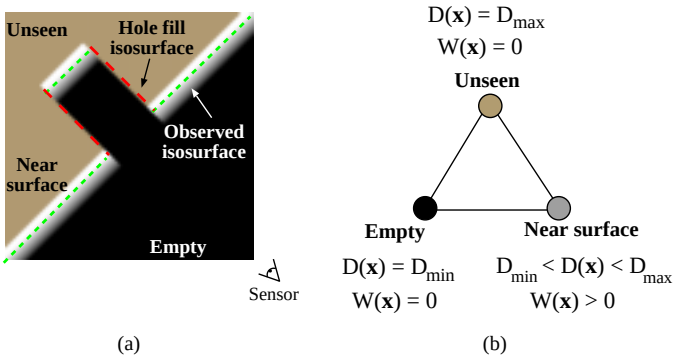


Figure 6. Volumetric grid with space carving and hole filling. (a) The regions in front of the surface are seen as empty, regions in the vicinity of the surface ramp through the zero-crossing, while regions behind remain unseen. The green (dashed) segments are the isosurfaces generated near the observed surface, while the red (dotted) segments are hole fillers, generated by tessellating over the transition from empty to unseen. In (b), we identify the three extremal voxel states with their corresponding function values.

We take advantage of this distinction when smoothing surfaces as described below.

Figure 6 illustrates the method for a single range image, and provides a diagram for the three-state classification scheme. The hole filler isosurfaces are "false" in that they are not representative of the observed surface, but they do derive from observed data. In particular, they correspond to a boundary that confines where the surface could plausibly exist. In practice, we find that many of these hole filler surfaces are generated in crevices that are hard for the sensor to reach.

Because the transition between unseen and empty is discontinuous and hole fill triangles are generated as an isosurface between these binary states, with no smooth transition, we generally observe aliasing artifacts in these areas. These artifacts can be eliminated by prefiltering the transition region before sampling on the voxel lattice using straightforward methods such as analytic filtering or super-sampling and averaging down. In practice, we have obtained satisfactory results by applying another technique: post-filtering the mesh after reconstruction using weighted averages of nearest vertex neighbors as described in [29]. The effect of this filtering step is to blur the hole fill surface. Since we know which triangles correspond to hole fillers, we need only concentrate the surface filtering on the these portions of the mesh. This localized filtering preserves the detail in the observed surface reconstruction. To achieve a smooth blend between filtered hole fill vertices and the neighboring "real" surface, we allow the filter weights to extend beyond and taper off into the vicinity of the hole fill boundaries.

We have just seen how "space carving" is a useful operation: it tells us much about the structure of free space, allowing us to fill holes in an intelligent way. However, our algorithm only carves back from observed surfaces. There are numerous situations where more carving would be useful. For example, the interior walls of a hollow cylinder may elude digitization, but by seeing through the hollow portion of the cylinder to a surface placed behind it, we can better approximate its geometry. We can extend the carving paradigm to cover these situations by placing such a backdrop behind the surfaces being scanned. By placing the backdrop outside of the voxel grid, we utilize it purely for carving space without introducing its geometry into the model.

5 Implementation

5.1 Hardware

The examples in this paper were acquired using a Cyberware 3030 MS laser stripe optical triangulation scanner. Figure 1b illustrates the scanning geometry: an object translates through a plane of laser light while the reflections are triangulated into depth profiles through a CCD camera positioned off axis. To improve the quality of the data, we apply the method of spacetime analysis as described in [6]. The benefits of this analysis include reduced range noise, greater immunity to reflectance changes, and less artifacts near range discontinuities.

When using traditional triangulation analysis implemented in hardware in our Cyberware scanner, the uncertainty in triangulation for our system follows the lines of sight of the expanding laser beam. When using the spacetime analysis, however, the uncertainty follows the lines of sight of the camera. The results described in section 6 of this paper were obtained with one or the other triangulation method. In each case, we adhere to the appropriate lines of sight when laying down signed distance and weight functions.

5.2 Software

The creation of detailed, complex models requires a large amount of input data to be merged into high resolution voxel grids. The examples in the next section include models generated from as many as 70 scans containing up to 12 million input vertices with volumetric grids ranging in size up to 160 million voxels. Clearly, time and space optimizations are critical for merging this data and managing these grids.

5.2.1 Run-length encoding

The core data structure is a run-length encoded (RLE) volume with three run types: empty, unseen, and varying. The varying fields are stored as a stream of varying data, rather than runs of constant value. Typical memory savings vary from 10:1 to 20:1. In fact, the space required to represent one of these voxel grids is usually *less* than the memory required to represent the final mesh as a list of vertices and triangle indices.

5.2.2 Fast volume traversal

Updating the volume from a range image may be likened to inverse volume rendering: instead of reading from a volume and writing to an image, we read from a range image and write to a volume. As a result, we leverage off of a successful idea from the volume rendering community: for best memory system performance, stream through the volume and the image simultaneously in scanline order [18]. In general, however, the scanlines of a range image are not aligned with the scanlines of the voxel grid, as shown in Figure 7a. By suitably resampling the range image, we obtain the desired alignment (Figure 7b). The resampling process consists of a depth rendering of the range surface using the viewing transformation specific to the lines of sight of the range sensor and using an image plane oriented to align with the voxel grid. We assign the weights as vertex "colors" to be linearly interpolated during the rendering step, an approach equivalent to Gouraud shading of triangle colors.

To merge the range data into the voxel grid, we stream through the voxel scanlines in order while stepping through the corresponding scanlines in the resampled range image. We map each voxel scanline to the correct portion of the range scanline as depicted in Figure 7d, and we resample the range data to yield a distance from the range surface. Using the combination rules given by equations 3 and 4, we update the run-length encoded structure. To preserve the linear memory structure of the RLE volume (and thus avoid using linked lists of runs scattered through the memory space), we read the voxel scanlines from the current volume and write the updated scanlines to a second RLE volume; i.e., we double-buffer the voxel grid. Note that depending on the scanner geometry, the mapping from voxels to range image pixels may not be linear, in which case care must be taken to resample appropriately [5].

For the case of merging range data only in the vicinity of the surface, we try to avoid processing voxels distant from the surface. To that end, we construct a binary tree of minimum and maximum depths for every adjacent pair of resampled range image scanlines. Before processing each voxel scanline, we query the binary tree to decide

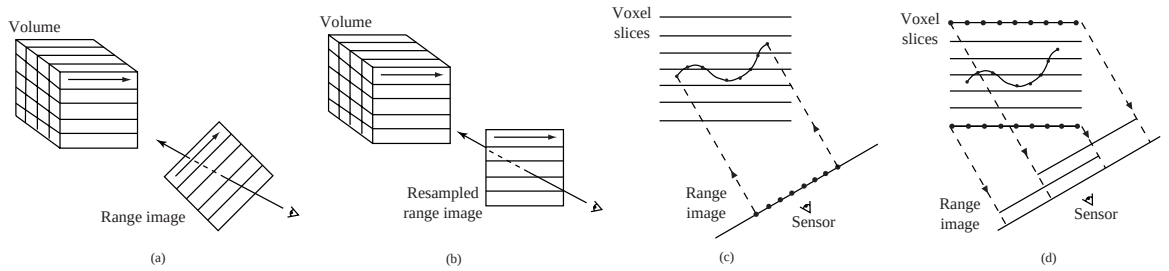


Figure 7. Range image resampling and scanline order voxel updates. (a) Range image scanlines are not in general oriented to allow for coherently streaming through voxel and range scanlines. (b) By resampling the range image, we can obtain the desired range scanline orientation. (c) Casting rays from the pixels on the range image means cutting across scanlines of the voxel grid, resulting in poor memory performance. (d) Instead, we run along scanlines of voxels, mapping them to the correct positions on the resampled range image.

which voxels, if any, are near the range surface. In this way, only relevant pieces of the scanline are processed. In a similar fashion, the space carving steps can be designed to avoid processing voxels that are not seen to be empty for a given range image. The resulting speedups from the binary tree are typically a factor of 15 without carving, and a factor of 5 with carving. We did not implement a brute-force volume update method, however we would expect the overall algorithm described here would be much faster by comparison.

5.2.3 Fast surface extraction

To generate our final surfaces, we employ a Marching Cubes algorithm [20] with a lookup table that resolves ambiguous cases [22]. To reduce computational costs, we only process voxels that have varying data or are at the boundary between empty and unseen.

6 Results

We show results for a number of objects designed to explore the robustness of our algorithm, its ability to fill gaps in the reconstruction, and its attainable level of detail. To explore robustness, we scanned a thin drill bit using the traditional method of optical triangulation. Due to the false edge extensions inherent in data from triangulation scanners [6], this particular object poses a formidable challenge, yet the volumetric method behaves robustly where the zippering method [30] fails catastrophically. The dragon sequence in Figure 11 demonstrates the effectiveness of carving space for hole filling. The use of a backdrop here is particularly effective in filling the gaps in the model. Note that we do not use the backdrop at all times, in part because the range images are much denser and more expensive to process, and also because the backdrop tends to obstruct the path of the object when automatically repositioning it with our motion control platform. Finally, the "Happy Buddha" sequence in Figure 12 shows that our method can be used to generate very detailed, hole-free models suitable for rendering and rapid manufacturing.

Statistics for the reconstruction of the dragon and Buddha models appear in Figure 8. With the optimizations described in the previous section, we were able to reconstruct the observed portions of the surfaces in under an hour on a 250 MHz MIPS R4400 processor. The space carving and hole filling algorithm is not completely optimized, but the execution times are still in the range of 3-5 hours, less than the time spent acquiring and registering the range images. For both models, the RMS distance between points in the original range images and points on the reconstructed surfaces is approximately 0.1 mm. This figure is roughly the same as the accuracy of the scanning technology, indicating a nearly optimal surface reconstruction.

7 Discussion and future work

We have described a new algorithm for volumetric integration of range images, leading to a surface reconstruction without holes. The algorithm has a number of desirable properties, including the representation of directional sensor uncertainty, incremental and order independent updating, robustness in the presence of sensor errors, and the ability to fill gaps in the reconstruction by carving space. Our use of a run-length encoded representation of the voxel grid and synchronized processing of voxel and resampled range image scanlines make the algorithm efficient. This in turn allows us to acquire and integrate a large number of range images. In particular, we demonstrate the ability to integrate up to 70 scans into a high resolution voxel grid to generate million polygon models in a few hours. These models are free of holes, making them suitable for surface fitting, rapid prototyping, and rendering.

There are a number of limitations that prevent us from generating models from an arbitrary object. Some of these limitations arise from the algorithm while others arise from the limitations of the scanning technology. Among the algorithmic limitations, our method has difficulty bridging sharp corners if no scan spans both surfaces meeting at the corner. This is less of a problem when applying our hole-filling algorithm, but we are also exploring methods that will work without hole filling. Thin surfaces are also problematic. As described in section 3, the influences of observed surfaces extend behind their estimated positions for each range image and can interfere with distance functions originating from scans of the opposite side of a thin surface. In this respect, the apexes of sharp corners also behave like thin surfaces. While we have limited this influence as much as possible, it still places a lower limit on the thickness of surface that we can reliably reconstruct without causing artifacts such as thickening of surfaces or rounding of sharp corners. We are currently working to lift this restriction by considering the estimated normals of surfaces.

Other limitations arise from the scanning technologies themselves. Optical methods such as the one we use in this paper can only provide data for external surfaces; internal cavities are not seen. Further, very complicated objects may require an enormous amount of scanning to cover the surface. Optical triangulation scanning has the additional problem that both the laser and the sensor must observe each point on the surface, further restricting the class of objects that can be scanned completely. The reflectance properties of objects are also a factor. Optical methods generally operate by casting light onto an object, but shiny surfaces can deflect this illumination, dark objects can absorb it, and bright surfaces can lead to interreflections. To minimize these effects, we often paint our objects with a flat, gray paint.

Straightforward extensions to our algorithm include improving the execution time of the space carving portion of the algorithm and demonstrating parallelization of the whole algorithm. In addition, more aggressive space carving may be possible by making inferences about sensor lines of sight that return no range data. In the future, we hope to apply our methods to other scanning technologies and to large scale objects such as terrain and architectural scenes.

Model	Scans	Input triangles	Voxel size (mm)	Volume dimensions	Exec. time (min)	Output triangles	Holes
Dragon	61	15 M	0.35	712x501x322	56	1.7 M	324
Dragon + fill	71	24 M	0.35	712x501x322	257	1.8 M	0
Buddha	48	5 M	0.25	407x957x407	47	2.4 M	670
Buddha + fill	58	9 M	0.25	407x957x407	197	2.6 M	0

Figure 8. Statistics for the reconstruction of the dragon and Buddha models, with and without space carving.

Acknowledgments

We would like to thank Phil Lacroute for his many helpful suggestions in designing the volumetric algorithms. Afra Zomorodian wrote the scripting interface for scanning automation. Homan Igehy wrote the fast scan conversion code, which we used for range image resampling. Thanks to Bill Lorensen for his marching cubes tables and mesh decimation software, and for getting the 3D hardcopy made. Matt Pharr did the accessibility shading used to render the color Buddha, and Pat Hanrahan and Julie Dorsey made helpful suggestions for RenderMan tricks and lighting models. Thanks also to David Addleman and George Dabrowski of Cyberware for their help and for the use of their scanner. This work was supported by the National Science Foundation under contract CCR-9157767 and Interval Research Corporation.

References

[1] C.L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *Proceedings of SIGGRAPH '95 (Los Angeles, CA, Aug. 6-11, 1995)*, pages 109–118. ACM Press, August 1995.

[2] J.-D. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, October 1984.

[3] C.H. Chien, Y.B. Sim, and J.K. Aggarwal. Generation of volume/surface octree from range data. In *The Computer Society Conference on Computer Vision and Pattern Recognition*, pages 254–60, June 1988.

[4] C. I. Connolly. Cumulative generation of octree models from range data. In *Proceedings, Intl. Conf. Robotics*, pages 25–32, March 1984.

[5] B. Curless. *Better optical triangulation and volumetric reconstruction of complex models from range images.* PhD thesis, Stanford University, 1996.

[6] B. Curless and M. Levoy. Better optical triangulation through spacetime analysis. In *Proceedings of IEEE International Conference on Computer Vision*, pages 987–994, June 1995.

[7] A. Dolenc. Software tools for rapid prototyping technologies in manufacturing. *Acta Polytechnica Scandinavica: Mathematics and Computer Science Series*, Ma62:1–111, 1993.

[8] D. Eberly, R. Gardner, B. Morse, S. Pizer, and C. Scharlach. Ridges for image analysis. *Journal of Mathematical Imaging and Vision*, 4(4):353–373, Dec 1994.

[9] H. Edelsbrunner and E.P. Mücke. Three-dimensional alpha shapes. In *Workshop on Volume Visualization*, pages 75–105, October 1992.

[10] A. Elfes and L. Matthies. Sensor integration for robot navigation: combining sonar and range data in a grid-based representation. In *Proceedings of the 26th IEEE Conference on Decision and Control*, pages 1802–1807, December 1987.

[11] H. Gagnon, M. Soucy, R. Bergevin, and D. Laurendeau. Registration of multiple range views for automatic 3-D model building. In *Proceedings 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 581–586, June 1994.

[12] E. Grosso, G. Sandini, and C. Frigato. Extraction of 3D information and volumetric uncertainty from multiple stereo images. In *Proceedings of the 8th European Conference on Artificial Intelligence*, pages 683–688, August 1988.

[13] P. Hebert, D. Laurendeau, and D. Poussart. Scene reconstruction and description: geometric primitive extraction from multiple viewed scattered data. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 286–292, June 1993.

[14] A. Hilton, A.J. Toddart, J. Illingworth, and T. Windeatt. Reliable surface reconstruction from multiple range images. In *Fourth European Conference on Com-*

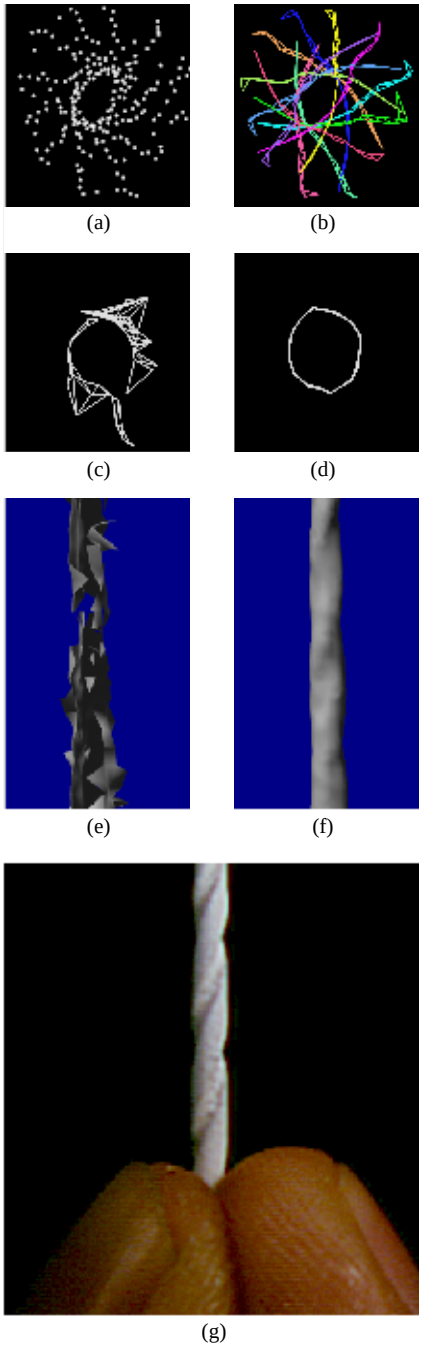


Figure 9. Merging range images of a drill bit. We scanned a 1.6 mm drill bit from 12 orientations at a 30 degree spacing using traditional optical triangulation methods. Illustrations (a) - (d) each show a plan (top) view of a slice taken through the range data and two reconstructions. (a) The range data shown as unorganized points: algorithms that operate on this form of data would likely have difficulty deriving the correct surface. (b) The range data shown as a set of wire frame tessellations of the range data: the false edge extensions pose a challenge to both polygon and volumetric methods. (c) A slice through the reconstructed surface generated by a polygon method: the zippering algorithm of Turk [31]. (d) A slice through the reconstructed surface generated by the volumetric method described in this paper. (e) A rendering of the zippered surface. (f) A rendering of the volumetrically generated surface. Note the catastrophic failure of the zippering algorithm. The volumetric method, however, produces a watertight model. (g) A photograph of the original drill bit. The drill bit was painted white for scanning.

puter Vision, volume I, pages 117–126, April 1996.

[15] Tsai-Hong Hong and M. O. Shneier. Describing a robot's workspace using a sequence of views from a moving camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(6):721–726, November 1985.

[16] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 71–78, July 1992.

[17] V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In these proceedings.

[18] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of SIGGRAPH '94 (Orlando, FL, July 24-29, 1994)*, pages 451–458. ACM Press, July 1994.

[19] A. Li and G. Crebbin. Octree encoding of objects from range images. *Pattern Recognition*, 27(5):727–739, May 1994.

[20] W.E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, July 1987.

[21] W.N. Martin and J.K. Aggarwal. Volumetric descriptions of objects from multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):150–158, March 1983.

[22] C. Montani, R. Scateni, and R. Scopigno. A modified look-up table for implicit disambiguation of marching cubes. *Visual Computer*, 10(6):353–355, 1994.

[23] M. Potmesil. Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing*, 40(1):1–29, October 1987.

[24] M. Rutishauser, M. Stricker, and M. Trobina. Merging range images of arbitrarily shaped objects. In *Proceedings 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 573–580, June 1994.

[25] M. Soucy and D. Laurendeau. A general surface approach to the integration of a set of range views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(4):344–358, April 1995.

[26] G. Succi, G. Sandini, E Grosso, and M. Tistarelli. 3D feature extraction from sequences of range data. In *Robotics Research. Fifth International Symposium*, pages 117–127, August 1990.

[27] R. Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Understanding*, 58(1):23–32, July 1993.

[28] G.H Tarbox and S.N. Gottschlich. IVIS: An integrated volumetric inspection system. In *Proceedings of the 1994 Second CAD-Based Vision Workshop*, pages 220–227, February 1994.

[29] G. Taubin. A signal processing approach to fair surface design. In *Proceedings of SIGGRAPH '95 (Los Angeles, CA, Aug. 6-11, 1995)*, pages 351–358. ACM Press, August 1995.

[30] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of SIGGRAPH '94 (Orlando, FL, July 24-29, 1994)*, pages 311–318. ACM Press, July 1994.

[31] Robert Weinstock. *The Calculus of Variations, with Applications to Physics and Engineering*. Dover Publications, 1974.

A Isosurface as least squares minimizer

It is possible to show that the isosurface of the weighted signed distance function is equivalent to a least squares minimization of squared distances between points on the range surfaces and points on the desired reconstruction. The key assumptions are that the range sensor is orthographic and that the range errors are independently distributed along sensor lines of sight. A full proof is beyond the scope of this paper, but we provide a sketch here. See [5] for details.

Consider a region, R, on the desired surface, f, which is observed by n range images. We define the error between an observed range surface and a possible reconstructed surface as the integral of the weighted squared distances between points on the range surface and the reconstructed surface. These distances are taken along the lines of sight of the sensor, commensurate with the predominant directions of uncertainty (see Figure 10). The total error is the sum of the integrals for the n range images:

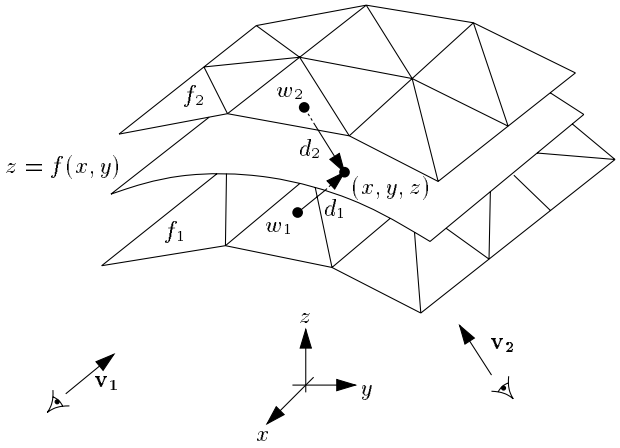


Figure 10. Two range surfaces, f_1 and f_2, are tessellated range images acquired from directions v_1 and v_2. The possible range surface, $z = f(x, y)$, is evaluated in terms of the weighted squared distances to points on the range surfaces taken along the lines of sight to the sensor. A point, (x, y, z), is shown here being evaluated to find its corresponding signed distances, d_1 and d_2, and weights, w_1 and w_2.

$$E(f) = \sum_{i=1}^{n} \iint_{A_i} w_i(s,t,f) d_i(s,t,f)^2 ds dt \tag{6}$$

where each (s, t) corresponds to a particular sensor line of sight for each range image, A_i is the domain of integration for the i'th range image, and $w_i(s, t, f)$ and $d_i(s, t, f)$ are the weights and signed distances taken along the i'th range image's lines of sight.

Now, consider a canonical domain, A, on a parameter plane, (x, y), over which R is a function $z = f(x, y)$. The total error can be rewritten as an integration over the canonical domain:

$$E(z) = \iint_A \sum_{i=1}^{n} \left[w_i(x,y,z) d_i(x,y,z)^2 \right] \left[\mathbf{v_i} \cdot (\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}, -1) \right] dx dy \tag{7}$$

where $\mathbf{v_i}$ is the sensing direction of the i'th range image, and the weights and distances are evaluated at each point, (x, y, z), by first mapping them to the lines of sight of the corresponding range image. The dot product represents a correction term that relates differential areas in A to differential areas in A_i. Applying the calculus of variations [31], we can construct a partial differential equation for the z that minimizes this integral. Solving this equation we arrive at the following relation:

$$\sum_{i=1}^{n} \partial_{\mathbf{v_i}} \left[w_i(x,y,z) d_i(x,y,z)^2 \right] = 0 \tag{8}$$

where $\partial_{\mathbf{v_i}}$ is the directional derivative along $\mathbf{v_i}$. Since the weight associated with a line of sight does not vary along that line of sight, and the signed distance has a derivative of unity along the line of sight, we can simplify this equation to:

$$\sum_{i=1}^{n} w_i(x,y,z) d_i(x,y,z) = 0 \tag{9}$$

This weighted sum of signed distances is the same as what we compute in equations 1 and 2, without the division by the sum of the weights. Since the this divisor is always positive, the isosurface we extract in section 3 is exactly the least squares minimizing surface described here.

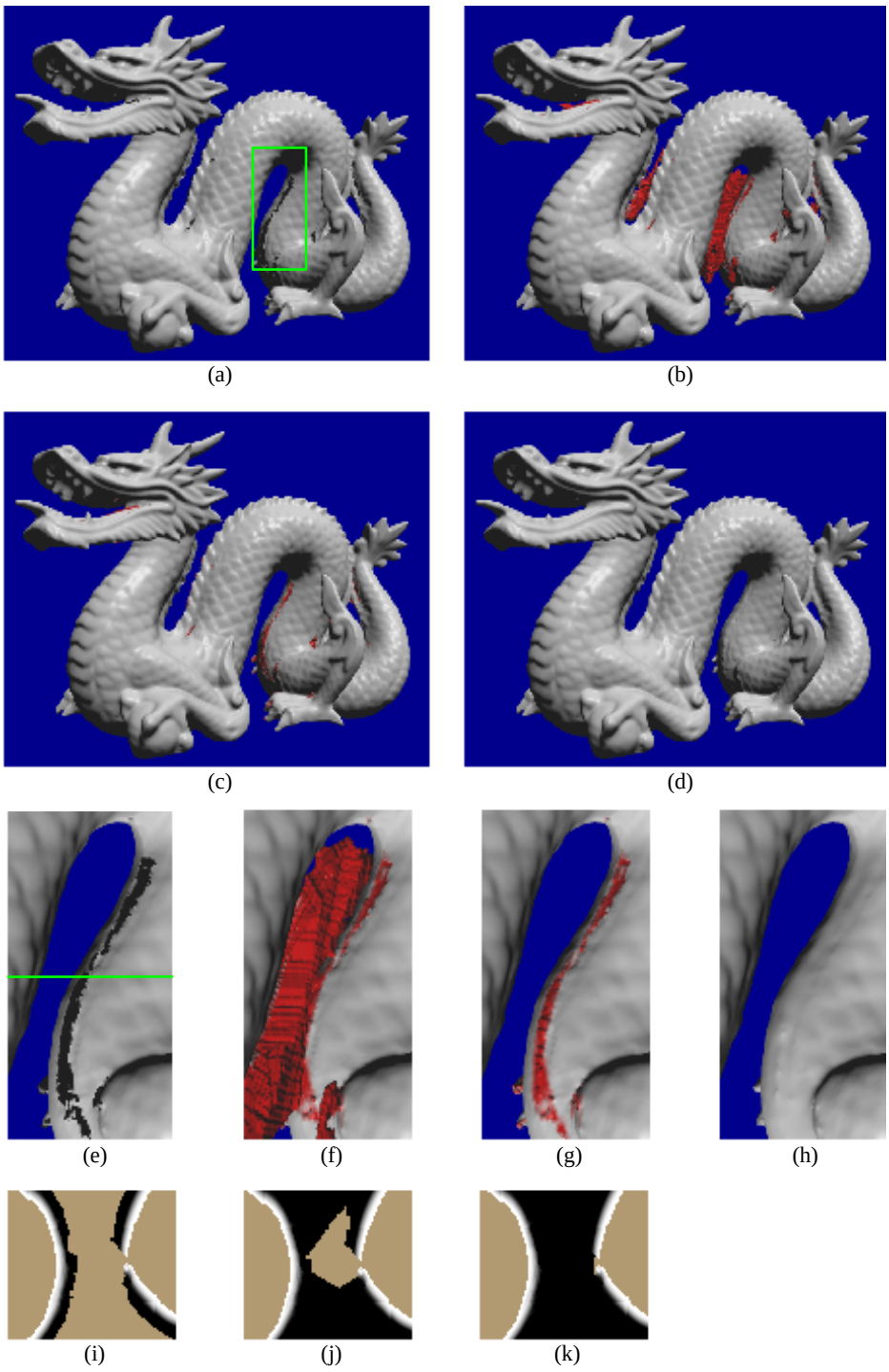
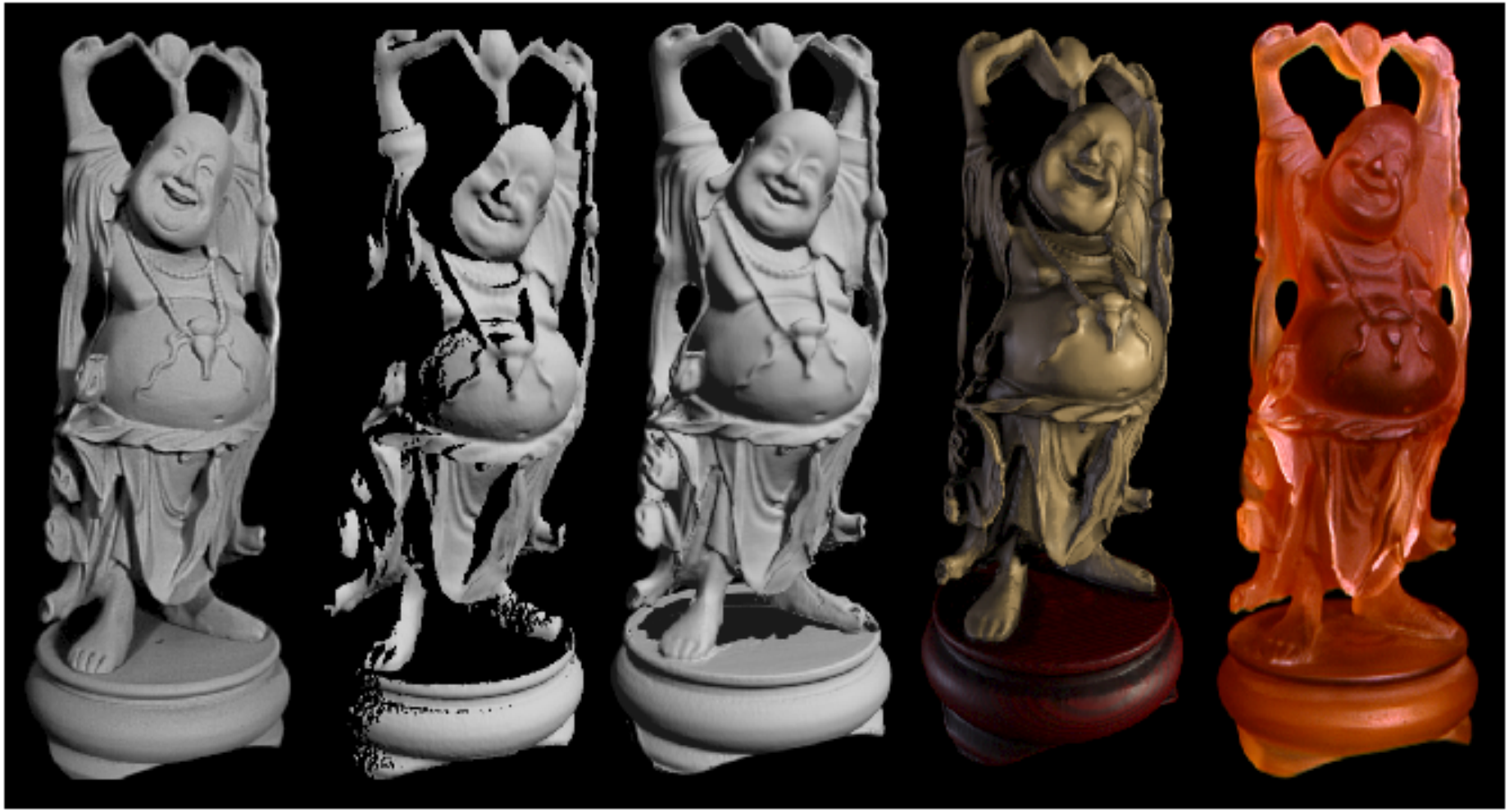


Figure 11. Reconstruction of a dragon. Illustrations (a) - (d) are full views of the dragon. Illustrations (e) - (h) are magnified views of the section highlighted by the green box in (a). Regions shown in red correspond to hole fill triangles. Illustrations (i) - (k) are slices through the corresponding volumetric grids at the level indicated by the green line in (e). (a)(e)(i) Reconstruction from 61 range images without space carving and hole filling. The magnified rendering highlights the holes in the belly. The slice through the volumetric grid shows how the signed distance ramps are maintained close to the surface. The gap in the ramps leads to a hole in the reconstruction. (b)(f)(j) Reconstruction with space carving and hole filling using the same data as in (a). While some holes are filled in a reasonable manner, some large regions of space are left untouched and create extraneous tessellations. The slice through the volumetric grid reveals that the isosurface between the unseen (brown) and empty (black) regions will be connected to the isosurface extracted from the distance ramps, making it part of the connected component of the dragon body and leaving us with a substantial number of false surfaces. (c)(g)(k) Reconstruction with 10 additional range images using "backdrop" surfaces to effect more carving. Notice how the extraneous hole fill triangles nearly vanish. The volumetric slice shows how we have managed to empty out the space near the belly. The bumpiness along the hole fill regions of the belly in (g) corresponds to aliasing artifacts from tessellating over the discontinuous transition between unseen and empty regions. (d)(h) Reconstruction as in (c)(g) with filtering of the hole fill portions of the mesh. The filtering operation blurs out the aliasing artifacts in the hole fill regions while preserving the detail in the rest of the model. Careful examination of (h) reveals a faint ridge in the vicinity of the smoothed hole fill. This ridge is actual geometry present in all of the renderings, (e)-(h). The final model contains 1.8 million polygons and is watertight.



(a) (b) (c) (d) (e)

Figure 12. Reconstruction and 3D hardcopy of the "Happy Buddha". The original is a plastic and rosewood statuette that stands 20 cm tall. Note that the camera parameters for each of these images is different, creating a slightly different perspective in each case. (a) Photograph of the original after spray painting it matte gray to simplify scanning. (b) Gouraud-shaded rendering of one range image of the statuette. Scans were acquired using a Cyberware scanner, modified to permit spacetime triangulation [6]. This figure illustrates the limited and fragmentary nature of the information available from a single range image. (c) Gouraud-shaded rendering of the 2.4 million polygon mesh after merging 48 scans, but before hole-filling. Notice that the reconstructed mesh has at least as much detail as the single range image, but is less noisy; this is most apparent around the belly. The hole in the base of the model corresponds to regions that were not observed directly by the range sensor. (d) RenderMan rendering of an 800,000 polygon decimated version of the hole-filled and filtered mesh built from 58 scans. By placing a backdrop behind the model and taking 10 additional scans, we were able to see through the space between the base and the Buddha's garments, allowing us to carve space and fill the holes in the base. (e) Photograph of a hardcopy of the 3D model, manufactured by 3D Systems, Inc., using stereolithography. The computer model was sliced into 500 layers, 150 microns apart, and the hardcopy was built up layer by layer by selectively hardening a liquid resin. The process took about 10 hours. Afterwards, the model was sanded and bead-blasted to remove the stair-step artifacts that arise during layered manufacturing.

Remeshing and Parameterization

Wim Sweldens

Subdivision

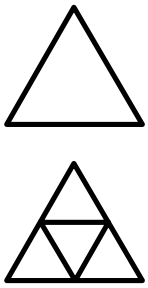
Critical tool for DGP

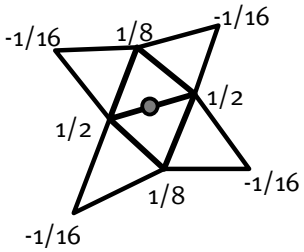
- Serves role of DSP low pass filter
- Currently used in variety of tools
- Basis of applications
  - Filtering, editing, compression

SUBDIVISION

Insertion of new vertices

- Linear combination of neighbors

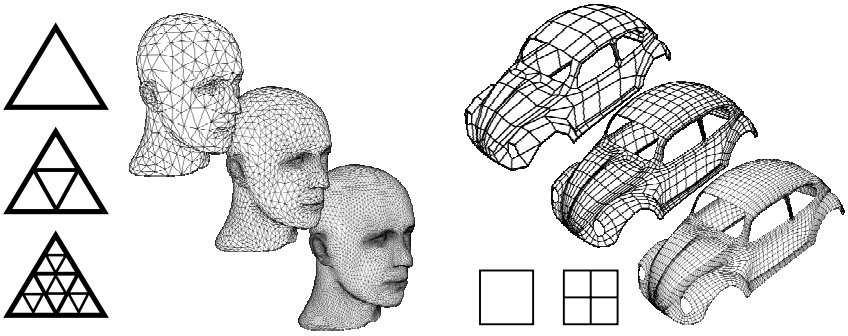




SUBDIVISION

Iterate insertion

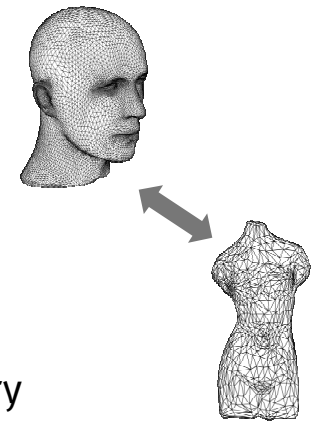
- Smooth, semi-regular mesh



Problem

Incompatible meshes

- Subdivision
  - Semi-regular mesh
  - Smooth geometry
- Scanned geometry
  - Irregular mesh
  - Non-smooth geometry



Solution

Connectivity

- Remeshing
- From irregular to regular mesh

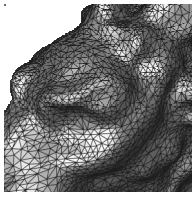
Geometry

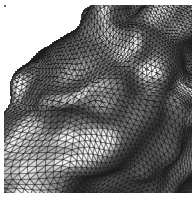
- Wavelets/details
- From smooth to non-smooth mesh

REMESHING

Change mesh

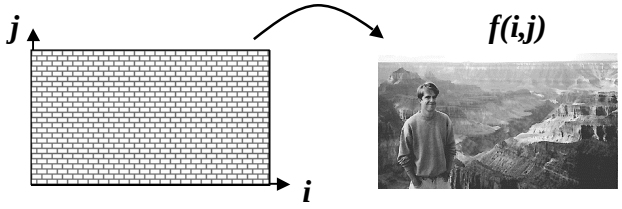
- Irregular mesh
- Semi-regular mesh
  - Coarse mesh
  - Quadrisection
- New vertex positions
  - Resampling





RESAMPLING

Well known for images/video

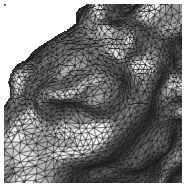


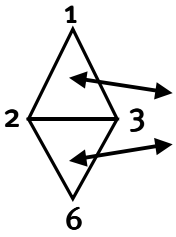
- Images are functions
  - Easy resampling

GEOMETRY DATA

Curved geometry

- Irregular sampling
- No functional description

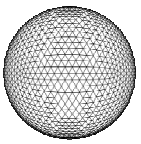




GEOMETRY

Not a function

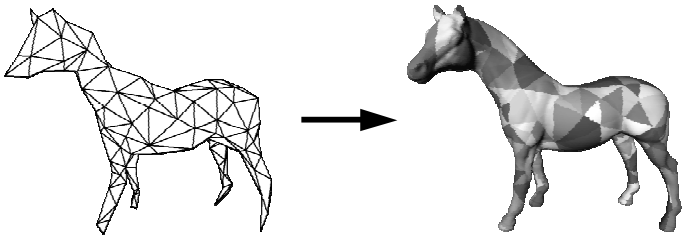
- Collection of vertex positions (x,y,z)



Parameterization

Global mathematical description

- bijection



domain parameterized surface

Literature

Eck et al. 95

- harmonic map remeshing

Krishnamurthy & Levoy 96

- spline patches under user control

Lee et al. 98

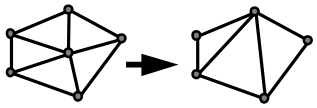
- MAPS

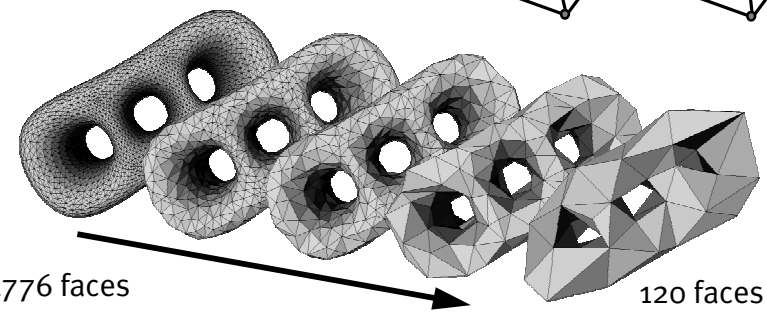
Guskov et al. 00

MAPS I

Create base domain

- mesh simplification

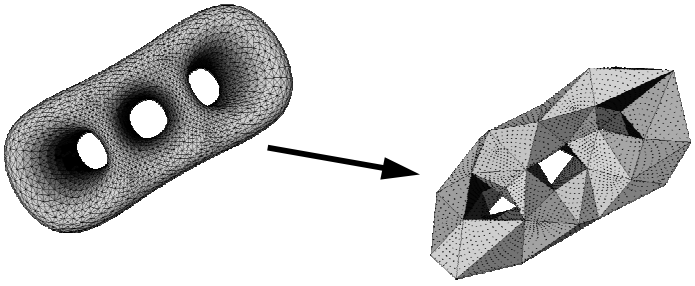




MAPS II

Construct mapping

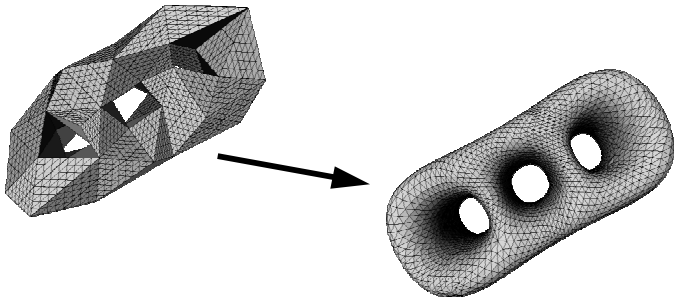
- surface to base domain



MAPS III

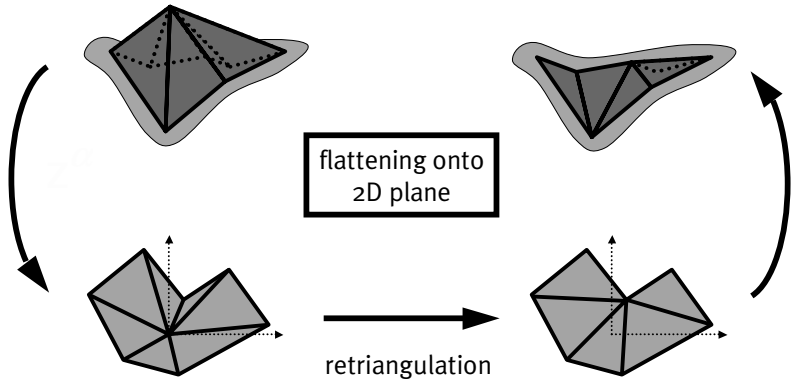
Construct parameterization

- globally smooth

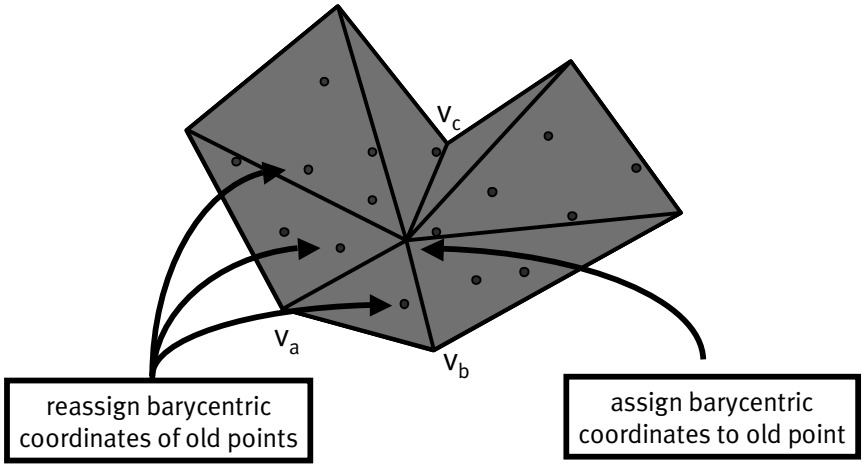


FLATTENING

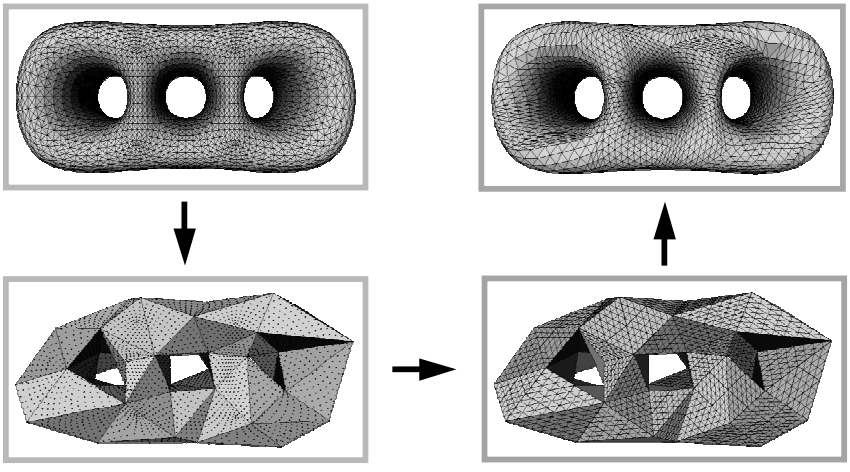
Local conformal mapping



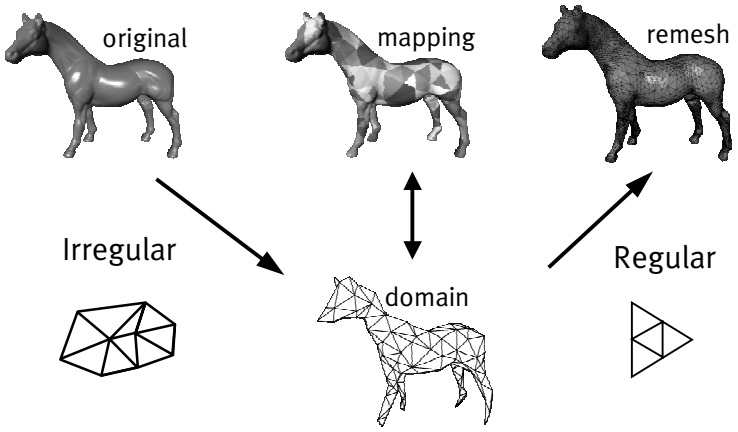
Parameter Assignment



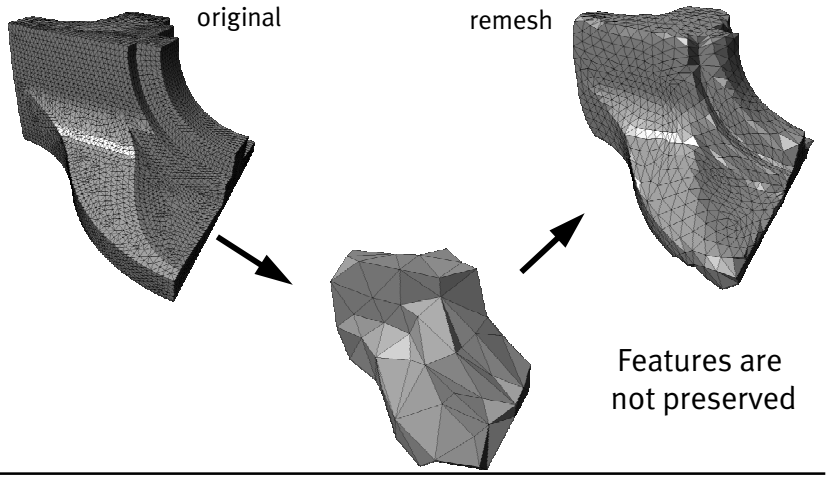
Parameterization



MAPS SUMMARY



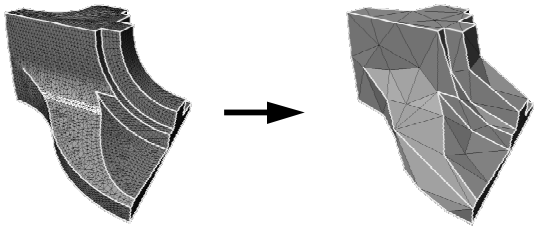
FEATURES!



Feature Preservation

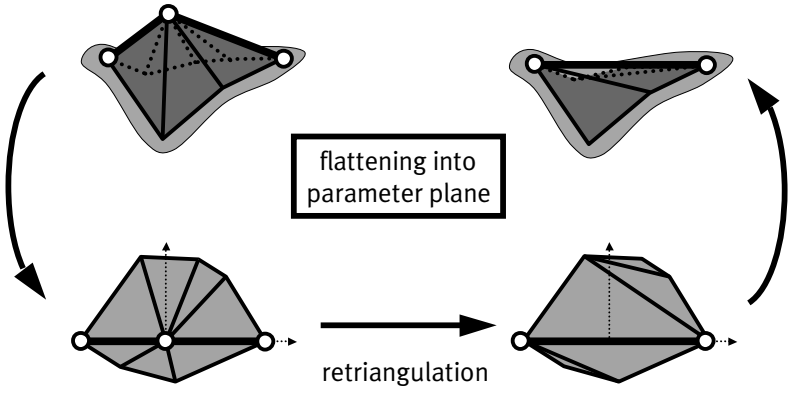
Algorithm:

- Find feature lines and points
- Mapping preserves features

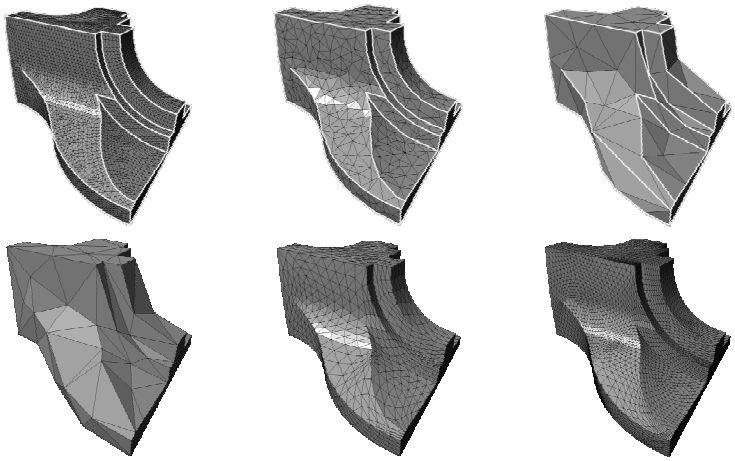


Feature preservation

Map feature lines to x-axis



Feature Preservation

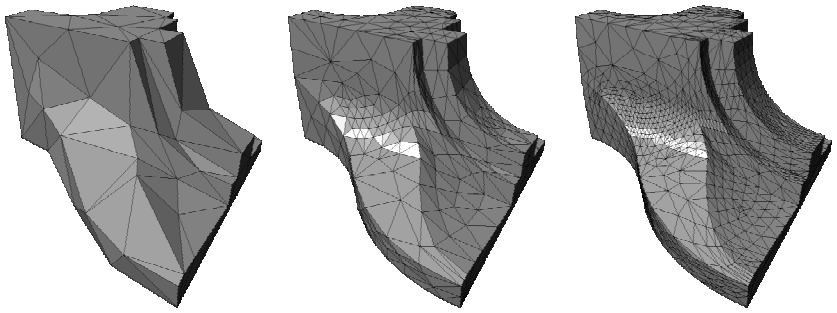


Adaptivity

Avoid uniform subdivision
- Exponential cost

Adaptive remeshing
- MAPS provides error criterium
  - Distance between remesh triangle and corresponding surface patch
  - Error driven adaptive refinement

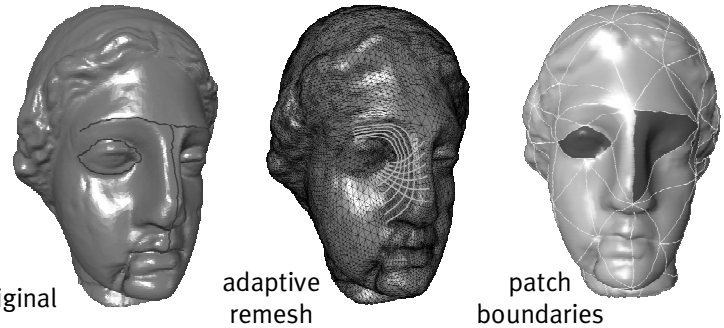
Adaptivity



5% 1%

Another Example

User constrained patch boundaries



original adaptive remesh patch boundaries

Direct Extraction

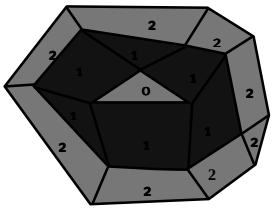
From volume to semi-regular

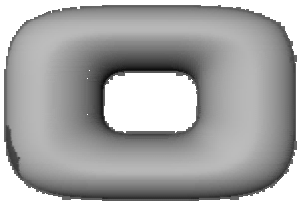
- No need to first run marching cubes
- McInerney, Terzopoulos, Qin, …
  - Problems with non-trivial topology
- Lachaud, Shinagawa, Stander, Hart
  - Uniqueness
- Here: Wood et al.

Topology Discovery

Isocontours of distance

- Ribbon construction
- Branch identification

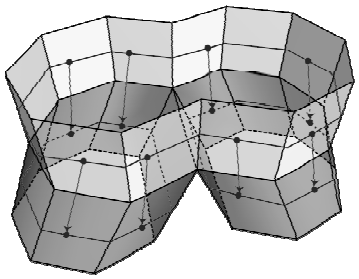




Topology Discovery

Topological graph

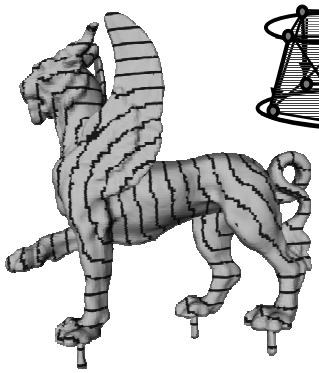
- Ribbon classification

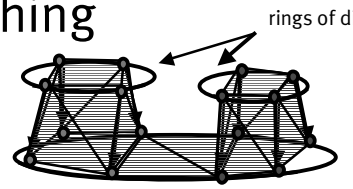


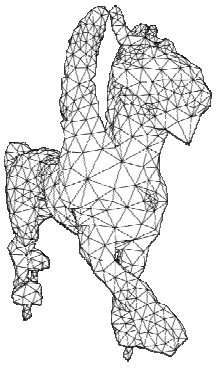
1-to-2 ribbon

Mesh Construction

Ribbon stitching







Coarse Mesh Refinement

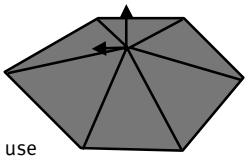
Construct hierarchical triangulation

- Match desired isosurface
- Good aspect ratios
- Smooth density variation
- Adaptive sampling density

Solver

Balloon model with forces

- External forces
- Internal forces
  - reparameterization force



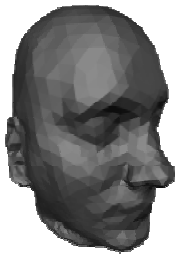
only use
tangential component

- Adaptive sampling

Results

MRI dataset 128^3

Coarse mesh construction: 0.5 seconds



# of triangles
MC: 58,684 Ours: 21,360

L^2 error $1.8*10^{-4}$

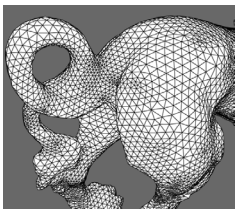
Results

3D Scanner 158x74x166

- non-trivial topology

Coarse mesh construction: 0.34 seconds

# of triangles
MC: 72,685 Ours: 46,996



L^2 error $3.3*10^{-4}$

MAPS: Multiresolution Adaptive Parameterization of Surfaces

Aaron W. F. Lee* Princeton University
Wim Sweldens† Bell Laboratories
Peter Schröder‡ Caltech
Lawrence Cowsar§ Bell Laboratories
David Dobkin¶ Princeton University

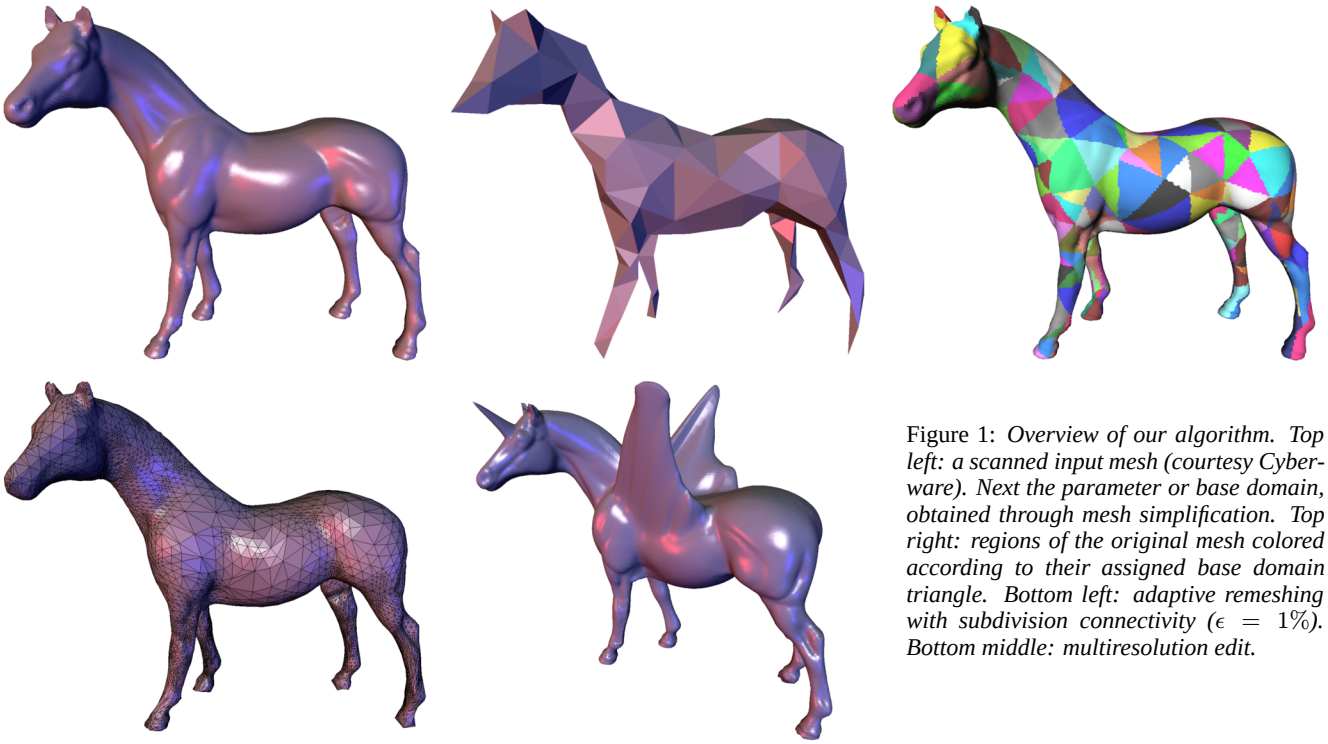


Figure 1: *Overview of our algorithm. Top left: a scanned input mesh (courtesy Cyberware). Next the parameter or base domain, obtained through mesh simplification. Top right: regions of the original mesh colored according to their assigned base domain triangle. Bottom left: adaptive remeshing with subdivision connectivity ($\epsilon = 1\%$). Bottom middle: multiresolution edit.*

Abstract

We construct smooth parameterizations of irregular connectivity triangulations of arbitrary genus 2-manifolds. Our algorithm uses hierarchical simplification to efficiently induce a parameterization of the original mesh over a base domain consisting of a small number of triangles. This initial parameterization is further improved through a hierarchical smoothing procedure based on Loop subdivision applied in the parameter domain. Our method supports both fully automatic and user constrained operations. In the latter, we accommodate point and edge constraints to force the alignment of iso-parameter lines with desired features. We show how to use the parameterization for fast, hierarchical subdivision connectivity remeshing with guaranteed error bounds. The remeshing algorithm constructs an adaptively subdivided mesh directly without first resorting to uniform subdivision followed by subsequent sparsification. It thus avoids the exponential cost of the latter. Our parameterizations are also useful for texture mapping and morphing applications, among others.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: *Picture/Image Generation – Display Algorithms, Viewing Algorithms*; I.3.5 [Computer Graphics]: *Computational Geometry and Object Modeling - Curve, Surface, Solid and Object Representations, Hierarchy and Geometric Transformations, Object Hierarchies.*

Additional Key Words and Phrases: Meshes, surface parameterization, mesh simplification, remeshing, texture mapping, multiresolution, subdivision surfaces, Loop scheme.

*wailee@cs.princeton.edu
†wim@bell-labs.com
‡ps@cs.caltech.edu
§cowsar@bell-labs.com
¶dpd@cs.princeton.edu

1 Introduction

Dense triangular meshes routinely result from a number of 3D acquisition techniques, e.g., laser range scanning and MRI volumetric imaging followed by iso-surface extraction (see Figure 1 top left). The triangulations form a surface of arbitrary topology—genus, boundaries, connected components—and have irregular connectivity. Because of their complex structure and tremendous size, these meshes are awkward to handle in such common tasks as storage, display, editing, and transmission.

Multiresolution representations are now established as a fundamental component in addressing these issues. Two schools exist. One approach extends classical multiresolution analysis and subdivision techniques to arbitrary topology surfaces [19, 20, 7, 3]. The alternative is more general and is based on sequential mesh simplification, e.g., progressive meshes (PM) [12]; see [11] for a review. In either case, the objective is to represent triangulated 2-manifolds in an efficient and flexible way, and to use this description in fast algorithms addressing the challenges mentioned above. Our approach fits in the first group, but draws on ideas from the second group.

An important element in the design of algorithms which manipulate mesh approximations of 2-manifolds is the construction of "nice" parameterizations when none are given. Ideally, the manifold is parameterized over a base domain consisting of a small number of triangles. Once a surface is understood as a function from the base domain into $\mathbf{R}^3$ (or higher-D when surface attributes are considered), many tools from areas such as approximation theory, signal processing, and numerical analysis are at our disposal. In particular, classical multiresolution analysis can be used in the design and analysis of algorithms. For example, error controlled, adaptive remeshing can be performed easily and efficiently. Figure 1 shows the outline of our procedure: beginning with an irregular input mesh (top left), we find a base domain through mesh simplification (top middle). Concurrent with simplification, a mapping is constructed which assigns every vertex from the original mesh to a base triangle (top right). Using this mapping an adaptive remesh with subdivision connectivity can be built (bottom left) which is now suitable for such applications as multiresolution editing (bottom middle). Additionally, there are other practical payoffs to good parameterizations, for example in texture mapping and morphing.

In this paper we present an algorithm for the fast computation of smooth parameterizations of dense 2-manifold meshes with arbitrary topology. Specifically, we make the following contributions

- We describe an $O(N \log N)$ time and storage algorithm to construct a logarithmic level hierarchy of arbitrary topology, irregular connectivity meshes based on the Dobkin-Kirkpatrick (DK) algorithm. Our algorithm accommodates geometric criteria such as area and curvature as well as vertex and edge constraints.
- We construct a smooth parameterization of the original mesh over the base domain. This parameterization is derived through repeated conformal remapping during graph simplification followed by a parameter space smoothing procedure based on the Loop scheme. The resulting parameterizations are of high visual and numerical quality.
- Using the smooth parameterization, we describe an algorithm for adaptive, hierarchical remeshing of arbitrary meshes into subdivision connectivity meshes. The procedure is fully automatic, but also allows for user intervention in the form of fixing point or path features in the original mesh. The remeshed manifold meets conservative approximation bounds.

Even though the ingredients of our construction are reminiscent of mesh simplification algorithms, we emphasize that our goal is not the construction of another mesh simplification procedure, but rather the construction of smooth parameterizations. We are particularly interested in using these parameterizations for remeshing, although they are useful for a variety of applications.

1.1 Related Work

A number of researchers have considered—either explicitly or implicitly—the problem of building parameterizations for arbitrary topology, triangulated surfaces. This work falls into two main categories: (1) algorithms which build a smoothly parameterized approximation of a set of samples (e.g. [14, 1, 17]), and (2) algorithms which remesh an existing mesh with the goal of applying classical multiresolution approaches [7, 8].

A related, though quite different problem, is the maintenance of a *given* parameterization during mesh simplification [4]. We emphasize that our goal is the *construction* of mappings when none are given.

In the following two sections, we discuss related work and contrast it to our approach.

1.1.1 Approximation of a Given Set of Samples

Hoppe and co-workers [14] describe a fully automatic algorithm to approximate a given polyhedral mesh with Loop subdivision patches [18] respecting features such as edges and corners. Their algorithm uses a non-linear optimization procedure taking into account approximation error and the number of triangles of the base domain. The result is a smooth parameterization of the original polyhedral mesh over the base domain. Since the approach only uses subdivision, small features in the original mesh can only be resolved accurately by increasing the number of triangles in the base domain accordingly. A similar approach, albeit using A-patches, was described by Bajaj and co-workers [1]. From the point of view of constructing parameterizations, the main drawback of algorithms in this class is that the number of triangles in the base domain depends heavily on the geometric complexity of the goal surface.

This problem was addressed in work of Krishnamurthy and Levoy [17]. They approximate densely sampled geometry with bicubic spline patches and displacement maps. Arguing that a fully automatic system cannot put iso-parameter lines where a skilled animator would want them, they require the user to lay out the entire network of top level spline patch boundaries. A coarse to fine matching procedure with relaxation is used to arrive at a high quality patch mesh whose base domain need not mimic small scale geometric features.

The principal drawback of their procedure is that the user is required to define the *entire* base domain rather then only selected features. Additionally, given that the procedure works from coarse to fine, it is possible for the procedure to "latch" onto the wrong surface in regions of high curvature [17, Figure 7].

1.1.2 Remeshing

Lounsbery and co-workers [19, 20] were the first to propose algorithms to extend classical multiresolution analysis to arbitrary topology surfaces. Because of its connection to the mathematical foundations of wavelets, this approach has proven very attractive (e.g. [22, 7, 27, 8, 3, 28]). The central requirement of these methods is that the input mesh have subdivision connectivity. This is generally not true for meshes derived from 3D scanning sources.

To overcome this problem, Eck and co-workers [7] developed an algorithm to compute smooth parameterizations of high resolution polyhedral meshes over a low face count base domain. Using such a mapping, the original surface can be remeshed using subdivision connectivity. After this conversion step, adaptive simplification, compression, progressive transmission, rendering, and editing become simple and efficient operations [3, 8, 28].

Eck et al. arrive at the base domain through a Voronoi tiling of the original mesh. Using a sequence of local harmonic maps, a parameterization which is smooth over each triangle in the base domain and which meets with C^0 continuity at base domain edges [7, Plate 1(f)] is constructed. Runtimes for the algorithm can be long because of the many harmonic map computations. This problem was recently addressed by Duchamp and co-workers [6], who reduced the harmonic map computations from their initial $O(N^2)$ complexity to $O(N \log N)$ through hierarchical preconditioning. The hier-

archy construction they employed for use in a multigrid solver is related to our hierarchy construction.

The initial Voronoi tile construction relies on a number of heuristics which render the overall algorithm fragile (for an improved version see [16]). Moreover, there is no explicit control over the number of triangles in the base domain or the placement of patch boundaries.

The algorithm generates only uniformly subdivided meshes which later can be decimated through classical wavelet methods. Many extra globally subdivided levels may be needed to resolve one small local feature; moreover, each additional level quadruples the amount of work and storage. This can lead to the intermediate construction of many more triangles than were contained in the input mesh.

1.2 Features of MAPS

Our algorithm was designed to overcome the drawbacks of previous work as well as to introduce new features. We use a fast coarsification strategy to define the base domain, avoiding the potential difficulties of finding Voronoi tiles [7, 16]. Since our algorithm proceeds from fine to coarse, correspondence problems found in coarse to fine strategies [17] are avoided, and all features are correctly resolved. We use conformal maps for continued remapping during coarsification to immediately produce a global parameterization of the original mesh. This map is further improved through the use of a hierarchical Loop smoothing procedure obviating the need for iterative numerical solvers [7]. Since the procedure is performed globally, derivative discontinuities at the edges of the base domain are avoided [7]. In contrast to fully automatic methods [7], the algorithm supports vertex and edge tags [14] to constrain the parameterization to align with selected features; however, the user is not required to specify the entire patch network [17]. During remeshing we take advantage of the original fine to coarse hierarchy to output a sparse, adaptive, subdivision connectivity mesh directly without resorting to a depth first oracle [22] or the need to produce a uniform subdivision connectivity mesh at exponential cost followed by wavelet thresholding [3].

2 Hierarchical Surface Representation

In this section we describe the main components of our algorithm, coarsification and map construction. We begin by fixing our notation.

2.1 Notation

When describing surfaces mathematically, it is useful to separate the topological and geometric information. To this end we introduce some notation adapted from [24]. We denote a triangular mesh as a pair $(\mathcal{P}, \mathcal{K})$, where $\mathcal{P}$ is a set of N point positions $p_i = (x_i, y_i, z_i) \in \mathbf{R}^3$ with $1 \leq i \leq N$, and $\mathcal{K}$ is an *abstract simplicial complex* which contains all the topological, i.e., adjacency information. The complex $\mathcal{K}$ is a set of subsets of $\{1, \ldots, N\}$. These subsets are called simplices and come in 3 types: vertices $v = \{i\} \in \mathcal{K}$, edges $e = \{i, j\} \in \mathcal{K}$, and faces $f = \{i, j, k\} \in \mathcal{K}$, so that any non-empty subset of a simplex of $\mathcal{K}$ is again a simplex of $\mathcal{K}$, e.g., if a face is present so are its edges and vertices.

Let e_i denote the standard i-th basis vector in $\mathbf{R}^N$. For each simplex s, its *topological realization* $|s|$ is the strictly convex hull of $\{e_i \mid i \in s\}$. Thus $|\{i\}| = e_i$, $|\{i, j\}|$ is the open line segment between e_i and e_j, and $|\{i, j, k\}|$ is an open equilateral triangle. The topological realization $|\mathcal{K}|$ is defined as $\cup_{s \in \mathcal{K}} |s|$. The *geometric realization* $\varphi(|\mathcal{K}|)$ relies on a linear map $\varphi : \mathbf{R}^N \to \mathbf{R}^3$ defined by $\varphi(e_i) = p_i$. The resulting polyhedron consists of points, segments, and triangles in $\mathbf{R}^3$.

Two vertices $\{i\}$ and $\{j\}$ are *neighbors* if $\{i, j\} \in \mathcal{K}$. A set of vertices is *independent* if no two vertices are neighbors. A set of vertices is *maximally independent* if no larger independent set contains it (see Figure 3, left side). The 1-ring neighborhood of a vertex $\{i\}$ is the set

$$\mathcal{N}(i) = \{j \mid \{i, j\} \in \mathcal{K}\}.$$

The *outdegree* K_i of a vertex is its number of neighbors. The *star* of a vertex $\{i\}$ is the set of simplices

$$\text{star}\,(i) = \bigcup_{i \in s,\, s \in \mathcal{K}} s.$$

We say that $|K|$ is a two dimensional manifold (or 2-manifold) with boundaries if for each i, $|\text{star}\,(i)|$ is homeomorphic to a disk (interior vertex) or half-disk (boundary vertex) in $\mathbf{R}^2$. An edge $e = \{i, j\}$ is called a *boundary edge* if there is only one face f with $e \subset f$.

We define a conservative curvature estimate, $\kappa(i) = |\kappa_1| + |\kappa_2|$ at p_i, using the principal curvatures κ_1 and κ_2. These are estimated by the standard procedure of first establishing a tangent plane at p_i and then using a second degree polynomial to approximate $\varphi(|\text{star}\,(i)|)$.

2.2 Mesh Hierarchies

An important part of our algorithm is the construction of a mesh hierarchy. The original mesh $(\mathcal{P}, \mathcal{K}) = (\mathcal{P}^L, \mathcal{K}^L)$ is successively simplified into a series of homeomorphic meshes $(\mathcal{P}^l, \mathcal{K}^l)$ with $0 \leq l < L$, where $(\mathcal{P}^0, \mathcal{K}^0)$ is the coarsest or base mesh (see Figure 4).

Several approaches for such mesh simplification have been proposed, most notably progressive meshes (PM) [12]. In PM the basic operation is the "edge collapse." A sequence of such atomic operations is prioritized based on approximation error. The linear sequence of edge collapses can be partially ordered based on topological dependence [25, 13], which defines levels in a hierarchy. The depth of these hierarchies appears "reasonable" in practice, though can vary considerably for the same dataset [13].

Our approach is similar in spirit, but inspired by the hierarchy proposed by Dobkin and Kirkpatrick (DK) [5], which guarantees that the number of levels L is $O(\log N)$. While the original DK hierarchy is built for convex polyhedra, we show how the idea behind DK can be used for general polyhedra. The DK atomic simplification step is a *vertex remove*, followed by a retriangulation of the hole.

The two basic operations "vertex remove" and "edge collapse" are related since an edge collapse into one of its endpoints corresponds to a vertex remove with a particular retriangulation of the resulting hole (see Figure 2). The main reason we chose an algorithm based on the ideas of the DK hierarchy is that it guarantees a logarithmic bound on the number of levels. However, we emphasize that the ideas behind our map constructions apply equally well to PM type algorithms.

2.3 Vertex Removal

One DK simplification step $\mathcal{K}^l \to \mathcal{K}^{l-1}$ consists of removing a maximally independent set of vertices with low outdegree (see Figure 3). To find such a set, the original DK algorithm used a greedy approach based only on *topological* information. Instead, we use a priority queue based on both *geometric and topological* information.

At the start of each level of the original DK algorithm, none of the vertices are marked and the set to be removed is empty. The

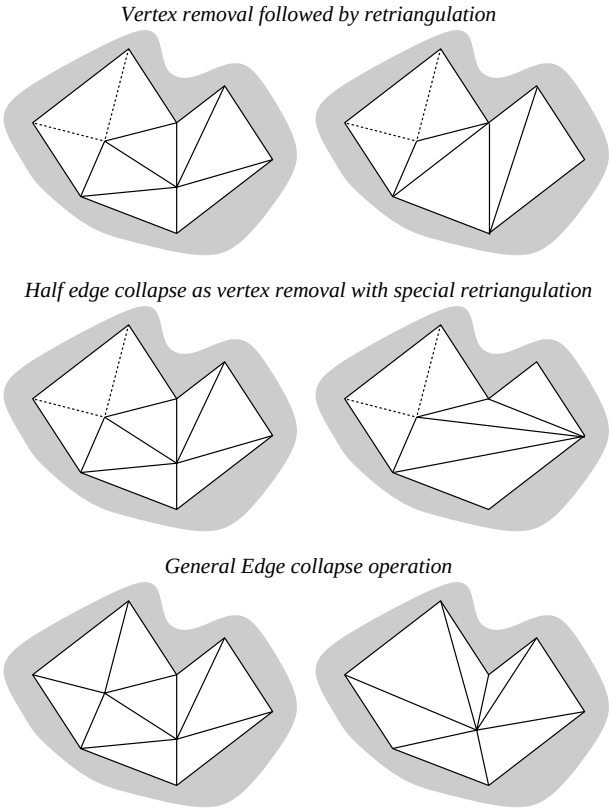


Figure 2: *Examples of different atomic mesh simplification steps. At the top vertex removal, in the middle half-edge collapse, and edge collapse at the bottom.*

algorithm randomly selects a non-marked vertex of outdegree less than 12, removes it and its star from $\mathcal{K}^l$, marks its neighbors as unremovable and iterates this until no further vertices can be removed. In a triangulated surface the average outdegree of a vertex is 6. Consequently, no more than half of the vertices can be of outdegree 12 or more. Thus it is guaranteed that at least $1/24$ of the vertices will be removed at each level [5]. In practice, it turns out one can remove roughly $1/4$ of the vertices reflecting the fact that the graph is four-colorable. Given that a constant fraction can be removed on each level, the number of levels behaves as $O(\log N)$. The entire hierarchy can thus be constructed in linear time.

In our approach, we stay in the DK framework, but replace the random selection of vertices by a priority queue based on geometric information. Roughly speaking, vertices with small and flat 1-ring neighborhoods will be chosen first. At level l, for a vertex $p_i \in \mathcal{P}^l$, we consider its 1-ring neighborhood $\varphi(|\text{star}\,(i)|)$ and compute its area $a(i)$ and estimate its curvature $\kappa(i)$. These quantities are computed relative to $\mathcal{K}^l$, the current level. We assign a priority to $\{i\}$ inversely proportional to a convex combination of relative area and curvature

$$w(\lambda, i) = \lambda \frac{a(i)}{\max_{p_i \in \mathcal{P}^l} a(i)} + (1-\lambda)\frac{\kappa(i)}{\max_{p_i \in \mathcal{P}^l} \kappa(i)}.$$

(We found $\lambda = 1/2$ to work well in our experiments.) Omitting all vertices of outdegree greater than 12 from the queue, removal of a constant fraction of vertices is still guaranteed. Because of the sort implied by the priority queue, the complexity of building the entire hierarchy grows to $O(N \log N)$.

Figure 4 shows three stages (original, intermediary, coarsest) of the DK hierarchy. Given that the coarsest mesh is homeomorphic to the original mesh, it can be used as the domain of a parameterization.

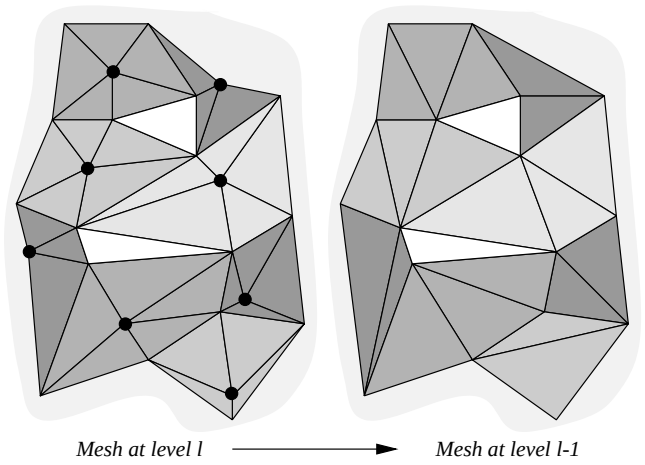


Figure 3: *On the left a mesh with a maximally independent set of vertices marked by heavy dots. Each vertex in the independent set has its respective star highlighted. Note that the star 's of the independent set do not tile the mesh (two triangles are left white). The right side gives the retriangulation after vertex removal.*

2.4 Flattening and Retriangulation

To find $\mathcal{K}^{l-1}$, we need to retriangulate the holes left by removing the independent set. One possibility is to find a plane into which to project the 1-ring neighborhood $\varphi(|\text{star}\,(i)|)$ of a removed vertex $\varphi(|i|)$ without overlapping triangles and then retriangulate the hole in that plane. However, finding such a plane, which may not even exist, can be expensive and involves linear programming [4].

Instead, we use the conformal map z^a [6] which minimizes metric distortion to map the neighborhood of a removed vertex into the plane. Let $\{i\}$ be a vertex to be removed. Enumerate cyclically the K_i vertices in the 1-ring $\mathcal{N}(i) = \{j_k \mid 1 \le k \le K_i\}$ such that $\{j_{k-1}, i, j_k\} \in \mathcal{K}^l$ with $j_0 = j_{K_i}$. A piecewise linear approximation of z^a, which we denote by μ_i, is defined by its values for the center point and 1-ring neighbors; namely, $\mu_i(p_i) = 0$ and $\mu_i(p_{j_k}) = r_k^a \exp(i\theta_k\, a)$, where $r_k = \|p_i - p_{j_k}\|$,

$$\theta_k = \sum_{l=1}^{k} \angle(p_{j_{l-1}}, p_i, p_{j_l}),$$

and $a = 2\pi/\theta_{K_i}$. The advantages of the conformal map are numerous: it always exists, it is easy to compute, it minimizes metric distortion, and it is a bijection and thus never maps two triangles on top of each other. Once the 1-ring is flattened, we can retriangulate the hole using, for example, a constrained Delaunay triangulation (CDT) (see Figure 5). This tells us how to build $\mathcal{K}^{l-1}$.

When the vertex to be removed is a boundary vertex, we map to a half disk by setting $a = \pi/\theta_{K_i}$ (assuming j_1 and j_{K_i} are boundary vertices and setting $\theta_1 = 0$). Retriangulation is again performed with a CDT.

3 Initial Parameterization

To find a parameterization, we begin by constructing a bijection Π from $\varphi(|\mathcal{K}^L|)$ to $\varphi(|\mathcal{K}^0|)$. The parameterization of the original mesh over the base domain follows from $\Pi^{-1}(\varphi(|\mathcal{K}^0|))$. In other words, the mapping of a point $p \in \varphi(|\mathcal{K}^L|)$ through Π is a point $p^0 = \Pi(v) \in \varphi(|\mathcal{K}^0|)$, which can be written as

$$p^0 = \alpha\, p_i + \beta\, p_j + \gamma\, p_k,$$

where $\{i, j, k\} \in \mathcal{K}^0$ is a face of the base domain and α, β and γ are barycentric coordinates, i.e., $\alpha + \beta + \gamma = 1$.

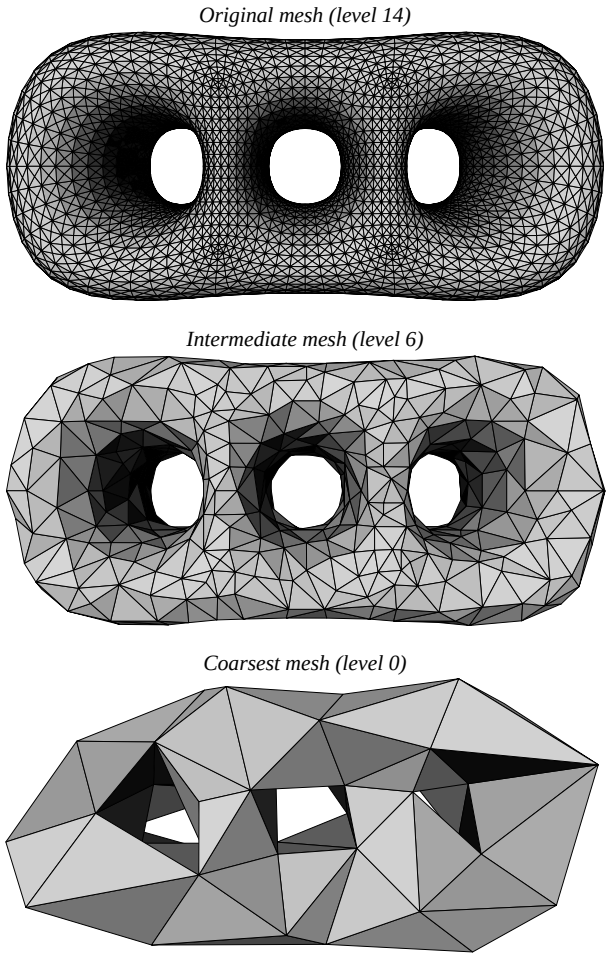


Figure 4: *Example of a modified DK mesh hierarchy. At the top the finest (original) mesh $\varphi(|\mathcal{K}^L|)$ followed by an intermediate mesh, and the coarsest (base) mesh $\varphi(|\mathcal{K}^0|)$ at the bottom (original dataset courtesy University of Washington).*

The mapping can be computed concurrently with the hierarchy construction. The basic idea is to successively compute piecewise linear bijections Π^l between $\varphi(|\mathcal{K}^L|)$ and $\varphi(|\mathcal{K}^l|)$ starting with Π^L, which is the identity, and ending with $\Pi^0 = \Pi$.

Notice that we only need to compute the value of Π^l at the vertices of $\mathcal{K}^L$. At any other point it follows from piecewise linearity.[1] Assume we are given Π^l and want to compute Π^{l-1}. Each vertex $\{i\} \in \mathcal{K}^L$ falls into one of the following categories:

1. $\{i\} \in \mathcal{K}^{l-1}$: The vertex is not removed on level l and survives on level $l-1$. In this case nothing needs to be done. $\Pi^{l-1}(p_i) = \Pi^l(p_i) = p_i$.

2. $\{i\} \in \mathcal{K}^l \setminus \mathcal{K}^{l-1}$: The vertex gets removed when going from l to $l-1$. Consider the flattening of the 1-ring around p_i (see Figure 5). After retriangulation, the origin lies in a triangle which corresponds to some face $t = \{j,k,m\} \in \mathcal{K}^{l-1}$ and has barycentric coordinates (α,β,γ) with respect to the vertices of that face, i.e., $\alpha\,\mu_i(p_j) + \beta\,\mu_i(p_k) + \gamma\,\mu_i(p_m)$ (see Figure 6). In that case, let $\Pi^{l-1}(p_i) = \alpha\,p_j + \beta\,p_k + \gamma\,p_m$.

3. $\{i\} \in \mathcal{K}^L \setminus \mathcal{K}^l$: The vertex was removed earlier, thus $\Pi^l(p_i) = \alpha'\,p_{j'} + \beta'\,p_{k'} + \gamma'\,p_{m'}$ for some triangle $t' = \{j',k',m'\} \in \mathcal{K}^l$. If $t' \in \mathcal{K}^{l-1}$, nothing needs to be done; otherwise, the independent set guarantees that exactly one vertex of t' is removed, say $\{j'\}$. Consider the conformal map $\mu_{j'}$ (Figure 6). After retriangulation, the $\mu_{j'}(p_i)$ lies in a triangle which corresponds to some face $t = \{j,k,m\} \in \mathcal{K}^{l-1}$ with barycentric coordinates (α,β,γ) (black dots within highlighted face in Figure 6). In that case, let $\Pi^{l-1}(p_i) = \alpha\,p_j + \beta\,p_k + \gamma\,p_m$ (i.e., all vertices in Figure 6 are reparameterized in this way).

[1] In the vicinity of vertices in $\mathcal{K}^l$ a triangle $\{i,j,k\} \in \mathcal{K}^L$ can straddle multiple triangles in $\mathcal{K}^l$. In this case the map depends on the flattening strategy used (see Section 2.4).

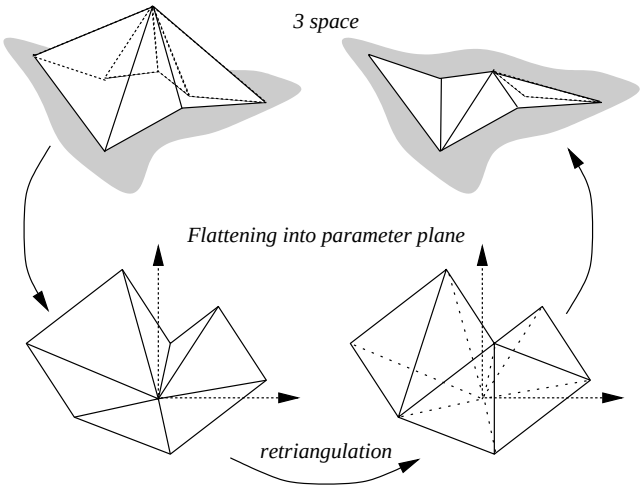


Figure 5: *In order to remove a vertex p_i, its star (i) is mapped from 3-space to a plane using the map z^a. In the plane the central vertex is removed and the resulting hole retriangulated (bottom right).*

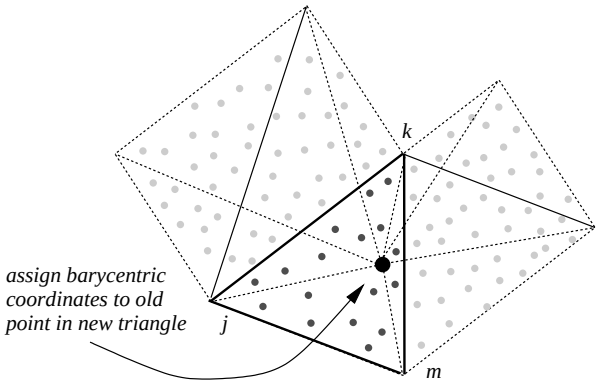


Figure 6: *After retriangulation of a hole in the plane (see Figure 5), the just removed vertex gets assigned barycentric coordinates with respect to the containing triangle on the coarser level. Similarly, all the finest level vertices that were mapped to a triangle of the hole now need to be reassigned to a triangle of the coarser level.*

Note that on every level, the algorithm requires a sweep through all the vertices of the finest level resulting in an overall complexity of $O(N \log N)$.

Figure 7 visualizes the mapping we just computed. For each point p_i from the original mesh, its mapping $\Pi(p_i)$ is shown with a dot on the base domain.

Caution: Given that every association between a 1-ring and its retriangulated hole is a bijection, so is the mapping Π. However, Π does not necessarily map a finest level triangle to a triangular region in the base domain. Instead the image of a triangle may be a non-convex region. In that case connecting the mapped vertices with straight lines can cause flipping, i.e., triangles may end up on

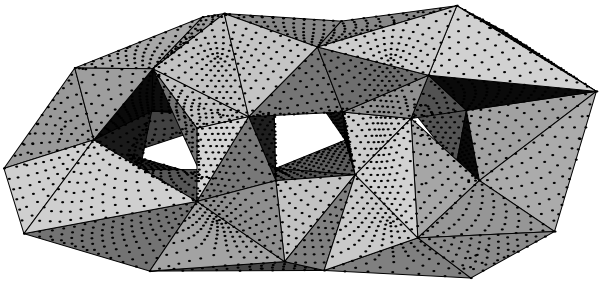


Figure 7: *Base domain $\varphi(|\mathcal{K}^0|)$. For each point p_i from the original mesh, its mapping $\Pi(p_i)$ is shown with a dot on the base domain.*

top of each other (see Figure 8 for an example). Two methods exist for dealing with this problem. First one could further subdivide the original mesh in the problem regions. Given that the underlying continuous map is a bijection, this is guaranteed to fix the problem. The alternative is to use some brute force triangle unflipping mechanism. We have found the following scheme to work well: adjust the parameter values of every vertex whose 2-neighborhood contains a flipped triangle, by replacing them with the averaged parameter values of its 1-ring neighbors [7].

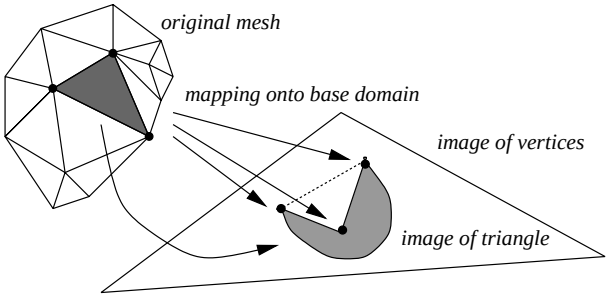


Figure 8: *Although the mapping Π from the original mesh to a base domain triangle is a bijection, triangles do not in general get mapped to triangles. Three vertices of the original mesh get mapped to a concave configuration on the base domain, causing the piecewise linear approximation of the map to flip the triangle.*

3.1 Tagging and Feature Lines

In the algorithm described so far, there is no *a priori* control over which vertices end up in the base domain or how they will be connected. However, often there are features which one wants to preserve in the base domain. These features can either be detected automatically or specified by the user.

We consider two types of features on the finest mesh: vertices and paths of edges. Guaranteeing that a certain vertex of the original mesh ends up in the base domain is straightforward. Simply mark that vertex as unremovable throughout the DK hierarchy.

We now describe an algorithm to guarantee that a certain path of edges on the finest mesh gets mapped to an edge of the base domain. Let $\{v_i \mid 1 \le i \le I\} \subset \mathcal{K}^L$ be a set of vertices on the finest level which form a path, i.e., $\{v_i, v_{i+1}\}$ is an edge. Tag all the edges in the path as feature edges. First tag v_1 and v_I, so called *dart points* [14], as unremovable so they are guaranteed to end up in the base domain. Let v_i be the first vertex on the interior of the path which gets marked for removal in the DK hierarchy, say, when going from level l to $l-1$. Because of the independent set property, v_{i-1} and v_{i+1} cannot be removed and therefore must belong to $\mathcal{K}^{l-1}$. When flattening the hole around v_i, tagged edges are treated like a boundary. We first straighten out the edges $\{v_{i-1}, v_i\}$ and

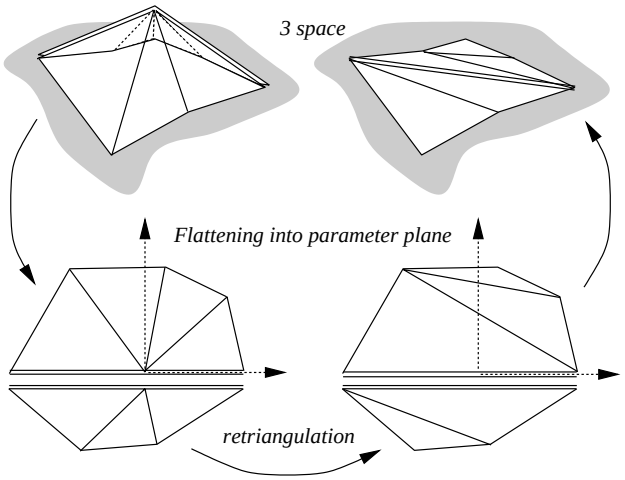


Figure 9: *When a vertex with two incident feature edges is removed, we want to ensure that the subsequent retriangulation adds a new feature edge to replace the two old ones.*

$\{v_i, v_{i+1}\}$ along the x-axis, and use two boundary type conformal maps to the half disk above and below (cf. the last paragraph of Section 2.4). When retriangulating the hole around v_i, we put the edge $\{v_{i-1}, v_{i+1}\}$ in $\mathcal{K}^{l-1}$, tag it as a feature edge, and compute a CDT on the upper and lower parts (see Figure 9). If we apply similar procedures on coarser levels, we ensure that v_1 and v_I remain connected by a path (potentially a single edge) on the base domain. This guarantees that Π maps the curved feature path onto the coarsest level edge(s) between v_1 and v_I.

In general, there will be multiple feature paths which may be closed or cross each other. As usual, a vertex with more than 2 incident feature edges is considered a corner, and marked as unremovable.

The feature vertices and paths can be provided by the user or detected automatically. As an example of the latter case, we consider every edge whose dihedral angle is below a certain threshold to be a feature edge, and every vertex whose curvature is above a certain threshold to be a feature vertex. An example of this strategy is illustrated in Figure 13.

3.2 A Quick Review

Before we consider the problem of remeshing, it may be helpful to review what we have at this point. We have established an initial bijection Π of the original surface $\varphi(|\mathcal{K}^L|)$ onto a base domain $\varphi(|\mathcal{K}^0|)$ consisting of a small number of triangles (e.g. Figure 7). We use a simplification hierarchy (Figure 4) in which the holes after vertex removal are flattened and retriangulated (Figures 5 and 9). Original mesh points get successively reparametrized over coarser triangulations (Figure 6). The resulting mapping is always a bijection; triangle flipping (Figure 8) is possible but can be corrected.

4 Remeshing

In this section, we consider remeshing using subdivision connectivity triangulations since it is both a convenient way to illustrate the properties of a parameterization and is an important subject in its own right. In the process, we compute a smoothed version of our initial parameterization. We also show how to efficiently construct an adaptive remeshing with guaranteed error bounds.

4.1 Uniform Remeshing

Since Π is a bijection, we can use Π^{-1} to map the base domain to the original mesh. We follow the strategy used in [7]: regularly (1:4) subdivide the base domain and use the inverse map to obtain a regular connectivity remeshing. This introduces a hierarchy of regular meshes $(\mathcal{Q}^m, \mathcal{R}^m)$ ($\mathcal{Q}$ is the point set and $\mathcal{R}$ is the complex) obtained from m-fold midpoint subdivision of the base domain $(\mathcal{P}^0, \mathcal{K}^0) = (\mathcal{Q}^0, \mathcal{R}^0)$. Midpoint subdivision implies that all new domain points lie *in* the base domain, $\mathcal{Q}^m \subset \varphi(|\mathcal{R}^0|)$ and $|\mathcal{R}^m| = |\mathcal{R}^0|$. All vertices of $\mathcal{R}^m \setminus \mathcal{R}^0$ have outdegree 6. The uniform remeshing of the original mesh on level m is given by $(\Pi^{-1}(\mathcal{Q}^m), \mathcal{R}^m)$.

We thus need to compute $\Pi^{-1}(q)$ where q is a point in the base domain with dyadic barycentric coordinates. In particular, we need to compute which triangle of $\varphi(|\mathcal{K}^L|)$ contains $\Pi^{-1}(q)$, or, equivalently, which triangle of $\Pi(\varphi(|\mathcal{K}^L|))$ contains q. This is a standard *point location* problem in an irregular triangulation. We use the point location algorithm of Brown and Faigle [2] which avoids looping that can occur with non-Delaunay meshes [10, 9]. Once we have found the triangle $\{i, j, k\}$ which contains q, we can write q as

$$q = \alpha\,\Pi(p_i) + \beta\,\Pi(p_j) + \gamma\,\Pi(p_k),$$

and thus

$$\Pi^{-1}(q) = \alpha\,p_i + \beta\,p_j + \gamma\,p_k \in \varphi(|\mathcal{K}^L|).$$

Figure 10 shows the result of this procedure: a level 3 uniform remeshing of a 3-holed torus using the Π^{-1} map.

A note on complexity: The point location algorithm is essentially a walk on the finest level mesh with complexity $O(\sqrt{N})$. Hierarchical point location algorithms, which have asymptotic complexity $O(\log N)$, exist [15] but have a much larger constant. Given that we schedule the queries in a systematic order, we almost always have an excellent starting guess and observe a constant number of steps. In practice, the finest level "walking" algorithm beats the hierarchical point location algorithms for all meshes we encountered (up to $100K$ faces).

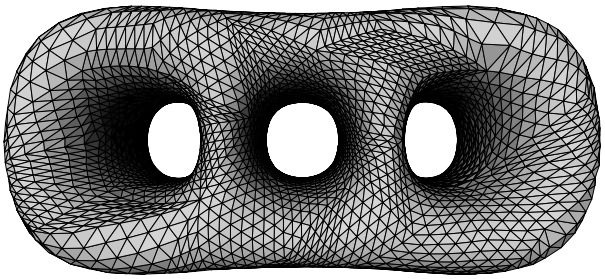


Figure 10: *Remeshing of 3 holed torus using midpoint subdivision. The parameterization is smooth within each base domain triangle, but clearly not across base domain triangles.*

4.2 Smoothing the Parameterization

It is clear from Figure 10 that the mapping we used is not smooth across global edges. One way to obtain global smoothness is to consider a map that minimizes a global smoothness functional and goes from $\varphi(|\mathcal{K}^L|)$ to $|\mathcal{K}^0|$ rather than to $\varphi(|\mathcal{K}^0|)$. This would require an iterative PDE solver. We have found computation of mappings to topological realizations that live in a high dimensional space to be needlessly cumbersome.

Instead, we use a much simpler and cheaper smoothing technique based on Loop subdivision. The main idea is to compute Π^{-1} at a smoothed version of the dyadic points, rather then at the dyadic points themselves (which can equivalently be viewed as changing the parameterization). To that end, we define a map $\mathcal{L}$ from the base domain to itself by the following modification of Loop:

- If all the points of the stencil needed for computing either a new point or smoothing an old point are inside the same triangle of the base domain, we can simply apply the Loop weights and the new points will be in that same face.
- If the stencil stretches across two faces of the base domain, we flatten them out using a "hinge" map at their common edge. We then compute the point's position in this flattened domain and extract the triangle in which the point lies together with its barycentric coordinates.
- If the stencil stretches across multiple faces, we use the conformal flattening strategy discussed earlier.

Note that the modifications to Loop force $\mathcal{L}$ to map the base domain onto the base domain. We emphasize that we do *not* apply the classic Loop scheme (which would produce a "blobby" version of the base domain). Nor are the surface approximations that we later produce Loop surfaces.

The composite map $\Pi^{-1} \circ \mathcal{L}$ is our *smoothed parameterization* that maps the base domain onto the original surface. The m-th level of uniform remeshing with the smoothed parameterization is $(\Pi^{-1} \circ \mathcal{L}(Q^m), \mathcal{R}^m)$, where Q^m, as before, are the dyadic points on the base domain. Figure 11 shows the result of this procedure: a level 3 uniform remeshing of a 3-holed torus using the smoothed parameterization.

When the mesh is tagged, we cannot apply smoothing across the tagged edges since this would break the alignment with the features. Therefore, we use modified versions of Loop which can deal with corners, dart points and feature edges [14, 23, 26] (see Figure 13).

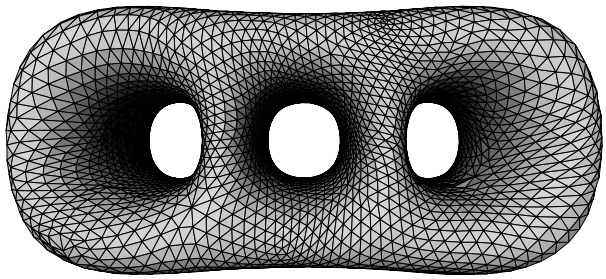


Figure 11: *The same remeshing of the 3-holed torus as in Figure 10, but this time with respect to a Loop smoothed parameterization.* **Note:** *Because the Loop scheme only enters in smoothing the* parameterization *the surface shown is still a sampling of the original mesh,* not *a Loop surface approximation of the original.*

4.3 Adaptive Remeshing

One of the advantages of meshes with subdivision connectivity is that classical multiresolution and wavelet algorithms can be employed. The standard wavelet algorithms used, e.g., in image compression, start from the finest level, compute the wavelet transform, and then obtain an efficient representation by discarding small wavelet coefficients. Eck et al. [7, 8] as well as Certain et al. [3] follow a similar approach: remesh using a uniformly subdivided grid followed by decimation through wavelet thresholding. This has the drawback that in order to resolve a small local feature on the original mesh, one may need to subdivide to a very fine level. Each extra

level quadruples the number of triangles, most of which will later be decimated using the wavelet procedure. Imagine, e.g., a plane which is coarsely triangulated except for a narrow spike. Making the spike width sufficiently small, the number of levels needed to resolve it can be made arbitrarily high.

In this section we present an algorithm which avoids first building a full tree and later pruning it. Instead, we immediately build the adaptive mesh with a guaranteed conservative error bound. This is possible because the DK hierarchy contains the information on how much subdivision is needed in any given area. Essentially, we let the irregular DK hierarchy "drive" the adaptive construction of the regular pyramid.

We first compute for each triangle $t \in \mathcal{K}^0$ the following error quantity:

$$E(t) = \max_{p_i \in \mathcal{P}^L \text{and } \Pi(p_i) \in \varphi(|t|)} \text{dist}(p_i, \varphi(|t|)).$$

This measures the distance between one triangle in the base domain and the vertices of the finest level mapped to that triangle.

The adaptive algorithm is now straightforward. Set a certain relative error threshold ϵ. Compute $E(t)$ for all triangles of the base domain. If $E(t)/B$, where B is the largest side of the bounding box, is larger than ϵ, subdivide the domain triangle using the Loop procedure above. Next, we need to reassign vertices to the triangles of level $m = 1$. This is done as follows: For each point $p_i \in \mathcal{P}^L$ consider the triangle t of $\mathcal{K}^0$ to which it it is currently assigned. Next consider the 4 children of t on level 1, t_j with $j = 0, 1, 2, 3$ and compute the distance between p_i and each of the $\varphi(|t_j|)$. Assign p_i to the closest child. Once the finest level vertices have been reassigned to level 1 triangles, the errors for those triangles can be computed. Now iterate this procedure until all triangles have an error below the threshold. Because all errors are computed from the finest level, we are guaranteed to resolve all features within the error bound. Note that we are not computing the true distance between the original vertices and a given approximation, but rather an easy to compute upper bound for it.

In order to be able to compute the Loop smoothing map $\mathcal{L}$ on an adaptively subdivided grid, the grid needs to satisfy a *vertex restriction criterion*, i.e., if a vertex has a triangle incident to it with depth i, then it must have a complete 1-ring at level $i-1$ [28]. This restriction may necessitate subdividing some triangles even if they are below the error threshold. Examples of adaptive remeshing can be seen in Figure 1 (lower left), Figure 12, and Figure 13.

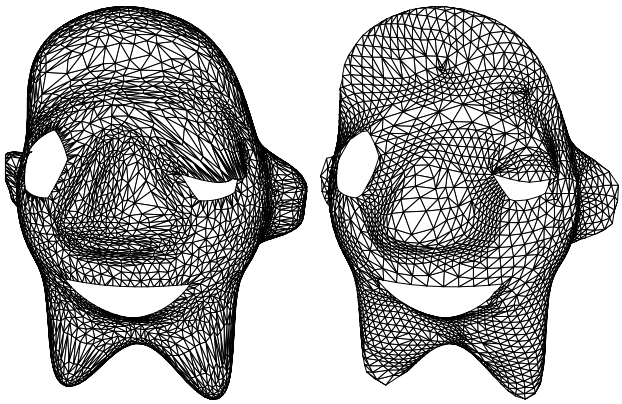


Figure 12: *Example remesh of a surface with boundaries.*

5 Results

We have implemented MAPS as described above and applied it to a number of well known example datasets, as well as some new ones. The application was written in C++ using standard computational geometry data structures, see e.g. [21], and all timings reported in this section were measured on a 200 MHz PentiumPro personal computer.

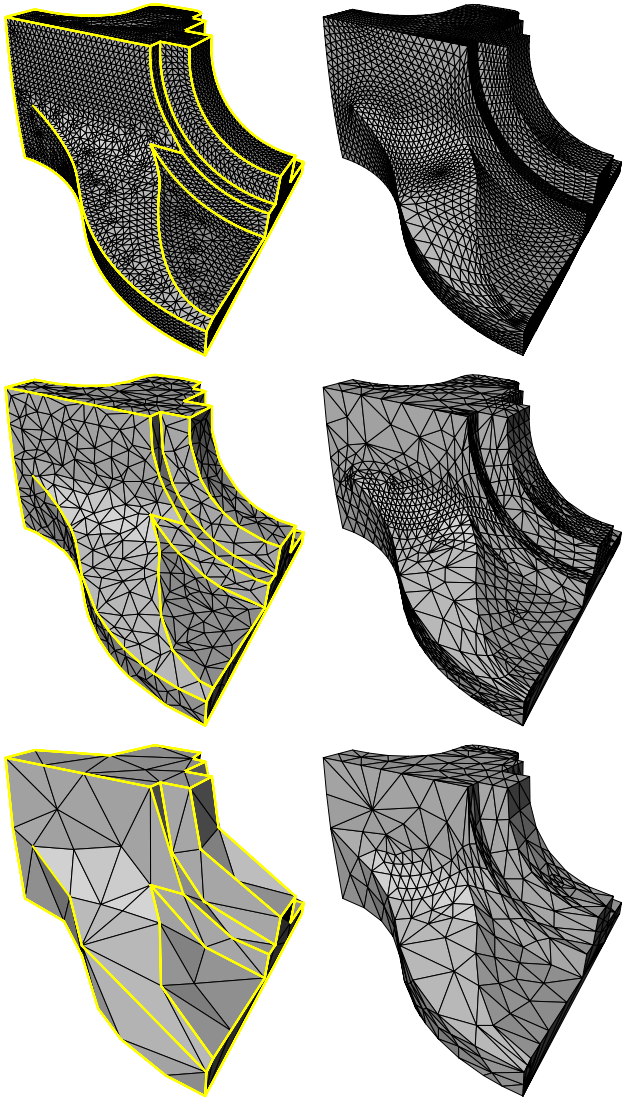


Figure 13: *Left (top to bottom): three levels in the DK pyramid, finest ($L = 15$) with 12946, intermediate ($l = 8$) with 1530, and coarsest ($l = 0$) with 168 triangles. Feature edges, dart and corner vertices survive on the base domain. Right (bottom to top): adaptive mesh with $\epsilon = 5\%$ and 1120 triangles (bottom), $\epsilon = 1\%$ and 3430 triangles (middle), and uniform level 3 (top). (Original dataset courtesy University of Washington.)*

The first example used throughout the text is the 3-holed torus. The original mesh contained 11776 faces. These were reduced in the DK hierarchy to 120 faces over 14 levels implying an average removal of 30% of the faces on a given level. The remesh of Figure 11 used 4 levels of uniform subdivision for a total of 30720 triangles.

The original sampled geometry of the 3-holed torus is smooth and did not involve any feature constraints. A more challenging case is presented by the fandisk shown in Figure 13. The original mesh (top left) contains 12946 triangles which were reduced to 168

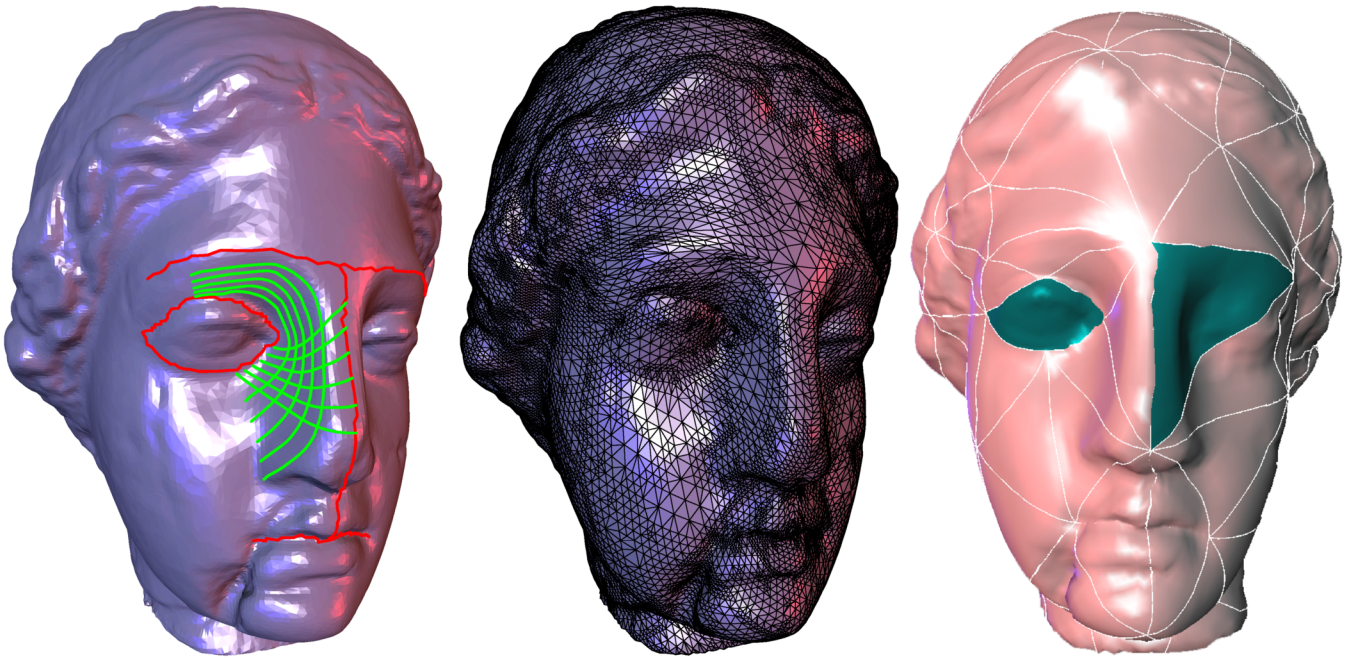


Figure 14: *Example of a constrained parameterization based on user input. Top: original input mesh (100000 triangles) with edge tags superimposed in red, green lines show some smooth iso-parameter lines of our parameterization. The middle shows an adaptive subdivision connectivity remesh. The bottom one patches corresponding to the eye regions (right eye was constrained, left eye was not) are highlighted to indicate the resulting alignment of top level patches with the feature lines. (Dataset courtesy Cyberware.)*

faces in the base domain over 15 levels (25% average face removal per level). The initial mesh had all edges with dihedral angles below 75^o tagged (1487 edges), resulting in 141 tagged edges at the coarsest level. Adaptive remeshing to within $\epsilon = 5\%$ and $\epsilon = 1\%$ (fraction of longest bounding box side) error results in the meshes shown in the right column. The top right image shows a uniform resampling to level 3, in effect showing iso-parameter lines of the parameterization used for remeshing. Note how the iso-parameter lines conform perfectly to the initially tagged features.

This dataset demonstrates one of the advantages of our method—inclusion of feature constraints—over the earlier work of Eck et al. [7]. In the original PM paper [12, Figure 12], Hoppe shows the simplification of the fandisk based on Eck's algorithm which does not use tagging. He points out that the multiresolution approximation is quite poor at low triangle counts and consequently requires many triangles to achieve high accuracy. The comparison between our Figure 13 and Figure 12 in [12] demonstrates that our multiresolution algorithm which incorporates feature tagging solves these problems.

Another example of constrained parameterization and subsequent adaptive remeshing is shown in Figure 14. The original dataset (100000 triangles) is shown on the left. The red lines indicate user supplied feature constraints which may facilitate subsequent animation. The green lines show some representative iso-parameter lines of our parameterization subject to the red feature constraints. Those can be used for computing texture coordinates. The middle image shows an adaptive subdivision connectivity remesh with 74698 triangles ($\epsilon = 0.5\%$). On the right we have highlighted a group of patches, 2 over the right (constrained) eye and 1 over the left (unconstrained) eye. This indicates how user supplied constraints force domain patches to align with desired features. Other enforced patch boundaries are the eyebrows, center of the nose, and middle of lips (see red lines in left image). This example illustrates how one places constraints like Krishnamurthy and Levoy [17]. We remove the need in their algorithms to specify the entire base domain. A user may want to control patch outlines for editing in one region (e.g., on the face), but may not care about what happens in other regions (e.g., the back of the head).

We present a final example in Figure 1. The original mesh (96966 triangles) is shown on the top left, with the adaptive, subdivision connectivity remesh on the bottom left. This remesh was subsequently edited in a interactive multiresolution editing system [28] and the result is shown on the bottom middle.

6 Conclusions and Future Research

We have described an algorithm which establishes smooth parameterizations for irregular connectivity, 2-manifold triangular meshes of arbitrary topology. Using a variant of the DK hierarchy construction, we simplify the original mesh and use piecewise linear approximations of conformal mappings to incrementally build a parameterization of the original mesh over a low face count base domain. This parameterization is further improved through a hierarchical smoothing procedure which is based on Loop smoothing in parameter space. The resulting parameterizations are of high quality, and we demonstrated their utility in an adaptive, subdivision connectivity remeshing algorithm that has guaranteed error bounds. The new meshes satisfy the requirements of multiresolution representations which generalize classical wavelet representations and are thus of immediate use in applications such as multiresolution editing and compression. Using edge and vertex constraints, the parameterizations can be forced to respect feature lines of interest without requiring specification of the entire patch network.

In this paper we have chosen remeshing as the primary application to demonstrate the usefulness of the parameterizations we pro-

Dataset	Input size (triangles)	Hierarchy creation	Levels	$\mathcal{P}^0$ size (triangles)	Remeshing tolerance	Remesh creation	Output size (triangles)
3-hole	11776	18 (s)	14	120	(NA)	8 (s)	30720
fandisk	12946	23 (s)	15	168	1%	10 (s)	3430
fandisk	12946	23 (s)	15	168	5%	5 (s)	1130
head	100000	160 (s)	22	180	0.5%	440 (s)	74698
horse	96966	163 (s)	21	254	1%	60 (s)	15684
horse	96966	163 (s)	21	254	0.5%	314 (s)	63060

Table 1: *Selected statistics for the examples discussed in the text. All times are in seconds on a 200 MHz PentiumPro.*

duce. The resulting meshes may also find application in numerical analysis algorithms, such as fast multigrid solvers. Clearly there are many other applications which benefit from smooth parameterizations, e.g., texture mapping and morphing, which would be interesting to pursue in future work. Because of its independent set selection the standard DK hierarchy creates topologically uniform simplifications. We have begun to explore how the selection can be controlled using geometric properties. Alternatively, one could use a PM framework to control geometric criteria of simplification. Perhaps the most interesting question for future research is how to incorporate topology changes into the MAPS construction.

Acknowledgments

Aaron Lee and David Dobkin were partially supported by NSF Grant CCR-9643913 and the US Army Research Office Grant DAAH04-96-1-0181. Aaron Lee was also partially supported by a Wu Graduate Fellowship and a Summer Internship at Bell Laboratories, Lucent Technologies. Peter Schröder was partially supported by grants from the Intel Corporation, the Sloan Foundation, an NSF CAREER award (ASC-9624957), a MURI (AFOSR F49620-96-1-0471), and Bell Laboratories, Lucent Technologies. Special thanks to Timothy Baker, Ken Clarkson, Tom Duchamp, Tom Funkhouser, Amanda Galtman, and Ralph Howard for many interesting and stimulation discussions. Special thanks also to Andrei Khodakovsky, Louis Thomas, and Gary Wu for invaluable help in the production of the paper. Our implementation uses the triangle facet data structure and code of Ernst Mücke.

References

[1] BAJAJ, C. L., BERNADINI, F., CHEN, J., AND SCHIKORE, D. R. Automatic Reconstruction of 3D CAD Models. Tech. Rep. 96-015, Purdue University, February 1996.

[2] BROWN, P. J. C., AND FAIGLE, C. T. A Robust Efficient Algorithm for Point Location in Triangulations. Tech. rep., Cambridge University, February 1997.

[3] CERTAIN, A., POPOVIĆ, J., DEROSE, T., DUCHAMP, T., SALESIN, D., AND STUETZLE, W. Interactive Multiresolution Surface Viewing. In *Computer Graphics (SIGGRAPH 96 Proceedings)*, 91–98, 1996.

[4] COHEN, J., MANOCHA, D., AND OLANO, M. Simplifying Polygonal Models Using Successive Mappings. In *Proceedings IEEE Visualization 97*, 395–402, October 1997.

[5] DOBKIN, D., AND KIRKPATRICK, D. A Linear Algorithm for Determining the Separation of Convex Polyhedra. *Journal of Algorithms 6* (1985), 381–392.

[6] DUCHAMP, T., CERTAIN, A., DEROSE, T., AND STUETZLE, W. Hierarchical Computation of PL harmonic Embeddings. Tech. rep., University of Washington, July 1997.

[7] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. In *Computer Graphics (SIGGRAPH 95 Proceedings)*, 173–182, 1995.

[8] ECK, M., AND HOPPE, H. Automatic Reconstruction of B-Spline Surfaces of Arbitrary Topological Type. In *Computer Graphics (SIGGRAPH 96 Proceedings)*, 325–334, 1996.

[9] GARLAND, M., AND HECKBERT, P. S. Fast Polygonal Approximation of Terrains and Height Fields. Tech. Rep. CMU-CS-95-181, CS Dept., Carnegie Mellon U., September 1995.

[10] GUIBAS, L., AND STOLFI, J. Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams. *ACM Transactions on Graphics 4*, 2 (April 1985), 74–123.

[11] HECKBERT, P. S., AND GARLAND, M. Survey of Polygonal Surface Simplification Algorithms. Tech. rep., Carnegie Mellon University, 1997.

[12] HOPPE, H. Progressive Meshes. In *Computer Graphics (SIGGRAPH 96 Proceedings)*, 99–108, 1996.

[13] HOPPE, H. View-Dependent Refinement of Progressive Meshes. In *Computer Graphics (SIGGRAPH 97 Proceedings)*, 189–198, 1997.

[14] HOPPE, H., DEROSE, T., DUCHAMP, T., HALSTEAD, M., JIN, H., MCDONALD, J., SCHWEITZER, J., AND STUETZLE, W. Piecewise Smooth Surface Reconstruction. In *Computer Graphics (SIGGRAPH 94 Proceedings)*, 295–302, 1994.

[15] KIRKPATRICK, D. Optimal Search in Planar Subdivisions. *SIAM J. Comput. 12* (1983), 28–35.

[16] KLEIN, A., CERTAIN, A., DEROSE, T., DUCHAMP, T., AND STUETZLE, W. Vertex-based Delaunay Triangulation of Meshes of Arbitrary Topological Type. Tech. rep., University of Washington, July 1997.

[17] KRISHNAMURTHY, V., AND LEVOY, M. Fitting Smooth Surfaces to Dense Polygon Meshes. In *Computer Graphics (SIGGRAPH 96 Proceedings)*, 313–324, 1996.

[18] LOOP, C. Smooth Subdivision Surfaces Based on Triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.

[19] LOUNSBERY, M. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. PhD thesis, Department of Computer Science, University of Washington, 1994.

[20] LOUNSBERY, M., DEROSE, T., AND WARREN, J. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *Transactions on Graphics 16*, 1 (January 1997), 34–73.

[21] MÜCKE, E. P. Shapes and Implementations in Three-Dimensional Geometry. Technical Report UIUCDCS-R-93-1836, University of Illinois at Urbana-Champaign, 1993.

[22] SCHRÖDER, P., AND SWELDENS, W. Spherical Wavelets: Efficiently Representing Functions on the Sphere. In *Computer Graphics (SIGGRAPH 95 Proceedings)*, Annual Conference Series, 1995.

[23] SCHWEITZER, J. E. *Analysis and Application of Subdivision Surfaces*. PhD thesis, University of Washington, 1996.

[24] SPANIER, E. H. *Algebraic Topology*. McGraw-Hill, New York, 1966.

[25] XIA, J. C., AND VARSHNEY, A. Dynamic View-Dependent Simplification for Polygonal Models. In *Proceedings Visualization 96*, 327–334, October 1996.

[26] ZORIN, D. *Subdivision and Multiresolution Surface Representations*. PhD thesis, California Institute of Technology, 1997.

[27] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interpolating Subdivision for Meshes with Arbitrary Topology. In *Computer Graphics (SIGGRAPH 96 Proceedings)*, 189–192, 1996.

[28] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interactive Multiresolution Mesh Editing. In *Computer Graphics (SIGGRAPH 97 Proceedings)*, 259–268, 1997.

Semi-Regular Mesh Extraction from Volumes

Zoë J. Wood, Caltech
Mathieu Desbrun, USC
Peter Schröder, Caltech
David Breen, Caltech

Abstract

We present a novel method to extract iso-surfaces from distance volumes. It generates high quality semi-regular multiresolution meshes of arbitrary topology. Our technique proceeds in two stages. First, a very coarse mesh with guaranteed topology is extracted. Subsequently an iterative multi-scale force-based solver refines the initial mesh into a semi-regular mesh with geometrically adaptive sampling rate and good aspect ratio triangles. The coarse mesh extraction is performed using a new approach we call *surface wavefront propagation*. A set of discrete iso-distance ribbons are rapidly built and connected while respecting the topology of the iso-surface implied by the data. Subsequent multi-scale refinement is driven by a simple force-based solver designed to combine good iso-surface fit and high quality sampling through reparameterization. In contrast to the Marching Cubes technique our output meshes adapt gracefully to the iso-surface geometry, have a natural multiresolution structure and good aspect ratio triangles, as demonstrated with a number of examples.

Keywords: Semi-regular meshes, subdivision, volumes, surface extraction, implicit functions, level set methods

1 Introduction

Iso-surface extraction is a fundamental technique of scientific visualization and one of the most useful tools for visualizing volume data. The predominant algorithm for iso-surface extraction, Marching Cubes (MC) [36], computes a local triangulation within each voxel of the volume containing the surface, resulting in a uniform resolution mesh. Often much smaller meshes adequately describe the surface since MC meshes tend to oversample the iso-surface, encumbering downstream applications, e.g., rendering, denoising, finite element simulations, and network transmission. These challenges can be addressed through multiresolution mesh representations.

We present a method for the *direct* extraction of an adaptively sampled multiresolution iso-surface mesh with good aspect ratio triangles. The multiresolution structure is based on adaptive *semi-regular* meshes, well known from the subdivision setting [54]. A semi-regular mesh consists of a coarsest level triangle mesh which is recursively refined through quadrisection. The resulting meshes have regular (valence 6) vertices almost everywhere. Adaptivity is achieved through terminating the recursion appropriately and enforcing a restriction criterion (triangles sharing an edge must be off by no more than one level of refinement). Conforming edges are used to prevent T-vertices (see Fig. 1). Because of their special structure such meshes enjoy many benefits including efficient compression [25] and editing [55] (among many others). Since the mesh hierarchy is represented through a forest of quad-trees, implementation is simple, elegant, and efficient. Figure 1 shows an example of a multiresolution semi-regular mesh extracted from a distance volume with our algorithm.

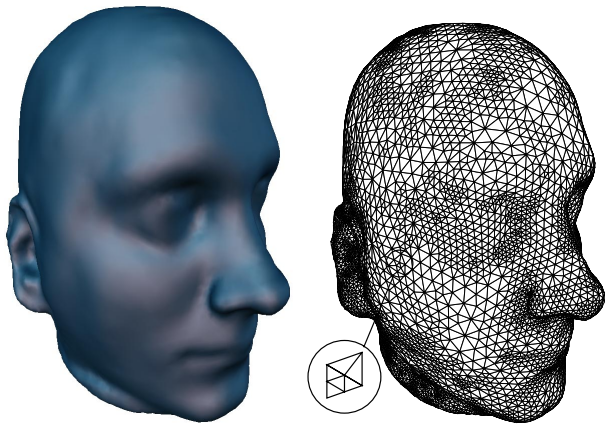


Figure 1: *Example extractions of adaptive semi-regular meshes from volumes using our algorithm.*

1.1 Contributions

We propose an algorithm for the extraction of semi-regular meshes directly from volume data. In a first step a coarse, irregular connectivity mesh with the same global topology as the iso-surface is extracted (Fig. 2, left). This stage works for arbitrary scalar volumes with well defined iso-surfaces and has a small memory footprint. In a second step the mesh is refined and its geometry optimized (Fig. 2, right). Here we require a distance volume for the desired iso-surface. During refinement, aspect ratios and sizes of triangles are controlled through adaptive quadrisection and *reparameterization forces*. Since our algorithm proceeds from coarser to finer resolutions, simple multi-scale methods are easily used. In particular we solve successively for the best fitting mesh at increasing resolutions using an upsampling of a coarser solution as the starting guess for the next finer level. In summary, novel aspects of our algorithm include:

- direct extraction of semi-regular meshes from volume data;
- a new and fast method to extract a topologically accurate coarse mesh with low memory requirements, suitable for large datasets;
- an improved force-based approach to quickly converge to a refined mesh that adaptively fits the data with good aspect ratio triangles.

1.2 Related Work

Traditional Methods and Multiresolution proceed by first constructing an MC mesh and then improving it through simplification [20] and/or remeshing [11, 29, 33, 28, 19]. Common mesh simplification algorithms have large memory footprints [21, 15] and are impractical for decimating meshes with millions of samples (see [35, 34] to address this issue). In addition, simplification algorithms create irregular connectivity meshes with non-smooth parameterizations. These cannot be compressed as efficiently as semi-regular meshes [25] leading to the need for remeshing. In

Figure 2: *Overview of our algorithm (left to right). Given a volume and a particular iso-value of interest a set of topologically faithful ribbons is constructed. Stitching them gives the coarsest level mesh for the solver. Adaptive refinement constructs a better and better fit with a semi-regular mesh.*

contrast we wish to directly extract multiresolution meshes with a smooth parameterization.

Alternatively multiresolution can be applied to the volume followed by subsequent MC extractions [50, 2]. Unfortunately, it is difficult to guarantee the topology of the mesh extracted from the simplified volume, e.g., small handles will disappear at various stages of the smoothing step, causing a change in the topology of the extracted mesh (see [16] for a new solution). In contrast our approach constructs a topologically accurate semi-regular mesh at every stage of the algorithm.

Deformable Model Approaches define the surface as the minimum (thin-plate) energy solution induced by a suitable potential function [40, 23, 38, 43, 28]. The second stage of our algorithm proceeds similarly with the important distinction that we exert specific control over the connectivity of the mesh to achieve a semi-regular structure and we use a balloon [5] approach coupled with a novel *reparameterization* force. Similar to previous approaches the initial mesh for our finite element solver must have the correct topology, however almost all previous approaches rely on user input to determine the appropriate global topology for the initial mesh [40, 43, 28, 38]. The largest advantage of our algorithm is our ability to extract a surface of arbitrary topology without any input from the user. Solvers which accommodate topological modifications are possible, but rather delicate [31, 39]. Instead we opt for a robust algorithm which *automatically* extracts a surface with the correct global topology from the volume data *without recourse to MC*.

Topological Graphs can be constructed to encode the topology of a surface. Our algorithm uses the adjacency relationships of the voxels in the volume to traverse the surface and record its connectivity in a graph that is topologically equivalent to the MC mesh for the same volume. This traversal and graph construction is related to work done by Lachaud [30] on topologically defined iso-surfaces. However, unlike Lachaud we do not triangulate the entire graph. Instead, our algorithm extracts a coarse mesh by eliminating redundant regions of the graph where the topology does not change.

Morse Theory and Reeb Graphs are also concerned with coding the topology of a surface [47, 45, 46]. However, neither method uniquely identifies the embedding of the surface in space, potentially leading to ambiguities in the topology coding. Work done on surface coding and Reeb graph construction by Shinagawa, using contours defined by a height function, resolves these ambiguities through requiring apriori knowledge [45, 46] of the number of handles. In contrast the topological graph we construct from the contours of the wavefront propagation *uniquely* determines the topology of the surface with *no* apriori information (for more details and a proof see [53]).

Distance Iso-contours are critical in our approach. We use ideas from level set methods on manifolds [26, 44] and discrete distance computations [32, 49]. Note that we compute these distances on implicitly defined (through the volume) surfaces, not on meshes. Specifically, we use the connectivity relationship of voxels in the volume to build a graph representing the surface. Distances are then propagated on this graph, creating a discrete distance graph. Iso-distance contours in this graph are used to correctly encode the topology of the surface without ever constructing an explicit mesh as in the MC algorithm.

Signed Distance Volumes are required by our solver, though the initial topology discovery stage runs on any volume with well-defined iso-contours. A signed distance volume stores the shortest signed distance to the surface at each voxel which is useful in a variety of applications [7, 6, 17, 42, 51]. Distance volumes are constructed by computing the shortest Euclidean distance within a narrow band around the desired iso-contour and then sweeping it out to the remaining voxels using a Fast Marching Method [44]. Distance volumes can easily be generated for a variety of input data. For example, distance volumes for MRI and CT data are computed by fitting a level set model to the desired iso-surface, creating a smooth segmentation of the input data [37, 52].

2 Coarse Mesh Extraction

In order to construct a topologically accurate coarse representation of a given iso-surface we slice the surface along contours that capture the overall topology. This concept is similar to representing a surface with a Reeb graph, which uses contours defined by a height function. The latter leads to ambiguities which we avoid by using contours of a distance function defined *on* the iso-surface. Examining the way these geometric contours are connected, we can always uniquely encode a topological graph of the iso-surface. This is achieved by discarding topologically redundant cross-sections, i.e., those where surface topology can not change.

Background Before we explain the details of this approach, recall some important theorems and definitions from Geometric Topology [41]. First, the topology of a 2-manifold M (closed polyhedral surface) is completely determined by its genus:

$$\chi(M) = V - E + F = 2(1 - g)$$

where χ is the Euler characteristic, V the number of vertices, E the number of edges, F the number of faces and g the genus. We use this fact and two related theorems:

- the Euler characteristic of an entire polyhedron can be decomposed into the sum of the Euler characteristics of smaller regions whose disjoint union is the polyhedron;
- the Euler characteristic of any given 2-manifold, or subset of a 2-manifold is invariant, *regardless of how the surface is triangulated*.

Given these facts, it is easy to see that topology can be captured accurately by selecting contours where the Euler characteristic of the associated region will change the genus of the surface. This selection is based on decomposing the surface into a combination of a few simple primitives:

1-sphere: A 1-sphere J is a set homeomorphic to a unit circle with $\chi(J) = 0$.

2-cell: A 2-cell D is a set homeomorphic to a disk with $\chi(D) = 1$.

For example, we can decompose a sphere into two 1-spheres (contours), two 2-cells (disks), and the triangulation between the two contours (which we call a *ribbon*) that respects the orientation of the original surface (see Fig. 3). Consider the combined Euler characteristic of these regions. As stated in the definitions, the Euler

characteristic of each of the two disks equals 1 while the Euler characteristic of the contours equals 0. Given this, and since the genus g of the sphere is 0, we deduce that the Euler characteristic χ of this ribbon is 0. This type of decomposition gives a general way

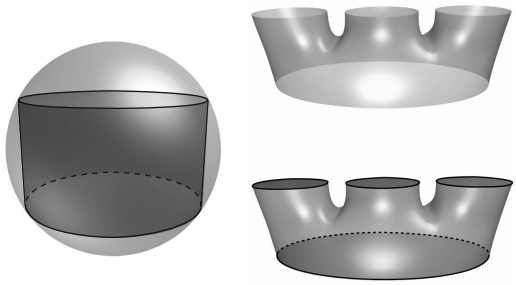


Figure 3: *On the left is a sphere decomposed into a ribbon and two disks. On the right (top) is a n-to-1 ribbon. On the right (bottom) is the closed ribbon, making it homeomorphic to a sphere*

to compute the Euler characteristic and thus the genus of a surface: separate the surface into regions that either are redundant or important with respect to the topology based on the Euler characteristic of those regions. It is important to note that we do not compute the Euler characteristic on a triangulated mesh and instead we rely on the implicit representation of the surface in the volume data.

Volume Setting Specifically, consider an implied surface intersected by a Cartesian grid. This intersection and the entire grid can be represented by tuples $(i, F(i))$, where i is a point in 3D space and $F(i)$ is the scalar value of the distance volume at that point in space. Without loss of generality we assume that the surface is the zero iso-contour of the volume. The surface will be pierced by the edges and faces of the Cartesian grid, creating a collection of patches each of which we denote as a *Surfel*, for surface element (Fig. 4, left). The edges of the grid which pierce the surface are denoted *active edges*. Their endpoints lie on opposite sides of the surface. Edge endpoints are considered either outside the surface if $F(i) \geq 0$, or inside the surface if $F(i) < 0$, thus edge endpoint cannot degenerately lie *on* the surface. The active edges intersect the surface at points called *nodes*. For the case of an iso-surface embedded in volume data, the resulting Surfel graph will be regular in the sense that all nodes are valence four. This Surfel graph is never triangulated, only its connectivity information is used to build the topological graph of the surface.

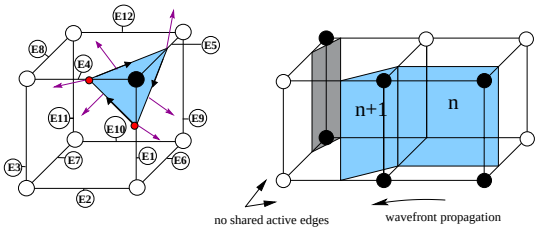


Figure 4: *Dark grey arrows indicate how to follow active edges from a given Surfel (left). On the right, the Surfel with distance n will propagate across its active edges the distance $n + 1$ to connected Surfels. Note that the other Surfel in this voxel will only receive a distance when the wavefront reaches it.*

Given this setting we return to the original goal of generating slices to subsample the surface while retaining the original topology. In order to code the Euler characteristic we traverse the Surfel graph and establish connectivity relationships between all the regions of the surface. Connectivity information is already implicitly represented by voxel adjacency in the volume. The construction of this graph has two parts. First we construct a distance tree, similar to propagating a wavefront across a surface in the geodesic setting. The frontier of the wavefront at any given distance will be a contour that geometrically fits the surface. Next we augment the distance tree by establishing connectivity between Surfels of the same distance, similar to constructing iso-contours for geodesics on the underlying iso-surface.

2.1 Wavefront Propagation and Distance Tree

The first step in our approach is to construct a topological distance tree by enumerating the Surfels through a wavefront-like propagation of Surfel distance. First consider the following graph representation of the surface: G is a graph, such that each vertex $s \in G$ is a Surfel and $n \in G$ is *1-node adjacent* to s if n shares a node with s. The edges of G are defined as the connections between each $s \in G$ and its 1-node adjacent neighbors. The distance tree D is induced by running Dijkstra's algorithm on G starting from any source Surfel s, with edge weights all equal to one. This propagates a distance [1] to all Surfels and constructs a tree such that:

- Each Surfel is 1-node adjacent to its parent in the tree;
- The shortest distance from a Surfel to the root is the depth of the Surfel in the tree hierarchy.

Surface Wavefront Propagation Any voxel that the surface passes through can serve as the root Surfel of our distance tree. From there, we construct the tree by enumerating the Surfels using Dijkstra's algorithm (Fig. 5, left). This propagation between adjacent Surfels can be done efficiently using active edges of the initial Cartesian grid to determine Surfel neighbors. The distance tree requires only a compact data structure and is represented by storing an additional integer and pointer per Surfel for each voxel as indicated by Figure 5(left). Each voxel typically has a single Surfel but up to four Surfels may be associated with a single voxel. This is of no consequence to the algorithm since we propagate the wavefront only across active edges (Fig. 4). Ambiguities can arise when using only the eight corners of a voxel to determine an ordering of the active edges but are easily avoided by selecting one consistent solution [3].

2.2 On-the-fly Construction of Topological Graph

The next step in the algorithm constructs a topological graph by augmenting the distance tree. This is done by collecting Surfels of the same distance into continuous ribbons, representing strips of the surface topology. The process of linking ribbons requires that we start with a given Surfel of distance n and traverse pairs of active edges—*faces* of the voxel bounding the given Surfel—in an ordered manner until we find another adjacent Surfel of the same distance n. As the ribbon is traversed, we enumerate an in-ribbon ordering for all the Surfels to assist in triangulation of the coarse mesh (see Fig. 4).

Constructing Ribbons To construct a consistent ordering within the ribbons, we use an idea very similar to work done on encoding a digital region boundary [13] and digital surface tracking [18]. Since the edges of each Surfel are ordered (see Fig. 4), a consistent traversal ordering can be established. For example, as shown in figure 4, this Surfel could be identified as: {E1, E4, E5}. During ribbon construction for the distance n, if this Surfel is reached by crossing the active edge pair {E1, E4}, first the next active edge pair {E4, E5} would be checked to see if the neighboring Surfel incident on this edge pair is the same distance. If it was not, the next pair would then be checked. One of these neighboring Surfels must be the same distance by definition of our wavefront propagation. The predecessor of the present Surfel must have at least one other successor which is 1-node adjacent to the present Surfel. This process of linking neighboring Surfels is continued

[1] When we refer to Surfel distance, we mean the path distance associated with the edges of G, i.e. each Surfel is distance 1 from its 1-node adjacent neighbors. This is a discrete, Surfel based distance.

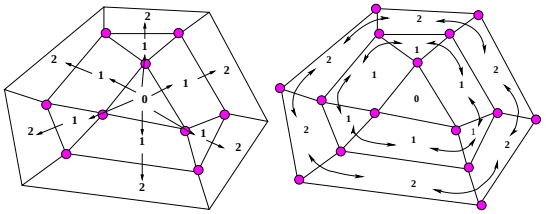


Figure 5: *Small portion of the distance tree overlayed on some Surfels (left). The Surfel labeled 0 is 1-node adjacent to all the Surfels labeled 1 since it shares at least one grey node with each of them. On the right is an example of 2-node adjacency between Surfels of the same distance as required in ribbon construction.*

until the initial Surfel of distance n is found, creating a continuous contour of the surface.

For a given distance n, after a single ribbon is constructed, we check to make sure that all the valid Surfels of distance n are part of a ribbon. If not, the ribbon construction is restarted with one of the unused Surfels at level n. This process continues until all Surfels are incorporated in the topological graph structure. Each distinct ribbon of the same distance is assigned a distinct branch name. Consequently, if there are multiple ribbons at level n, they will have unique branch names, either derived from their parent or assigned uniquely for completely new branches.

Cleanup of Ribbons If distance is propagated naïvely, ribbons could have tails (Fig. 7). Tails are large or small dead-ends of the wavefront. A dead-end of a wave front occurs when the wavefront runs into itself. Tails do not provide additional topological information [53] and are removed by pruning them from the distance tree during distance propagation: if a voxel cannot propagate its distance forward because all of its neighbors have already been visited, it is pruned from the distance tree.

The Topological Graph This construction guarantees that the topological graph has particular properties. Specifically, our topological graph is a representation of all the Surfels such that:

- All of the properties of a distance tree hold;
- Every Surfel has 2-node adjacency with exactly two other Surfels of the graph that are of the same distance and the same branch number — i.e. they share an edge (see Fig. 5, right).

These criteria establish that our topological graph is essentially composed of a collection of continuous contours of the surface. The dual of these contours are homeomorphic to a 1-sphere and combined with the root Surfel and leaf ribbons (homeomorphic to 2-cells), can be used to completely code the topology of the surface.

2.3 Coarse Mesh Construction

The topological graph provides everything needed to build the coarse mesh. In order to have a good coarse sampling of the surface, we only include the smallest number of ribbons necessary: Ribbons essential for coding topology are those inducing topological *events*. A ribbon represents a topological event only if it contributes to a change in the Euler characteristic of that region of the surface.

Ribbon Classification Consider the Euler characteristic of the three types of ribbon adjacencies:

Endcaps: A root Surfel or a leaf ribbon: these are 2-cells with $\chi = 1$.

1-to-1 ribbon : The most common case for a ribbon comprised of two connected 1-spheres with $\chi = 0$ (by the same argument used in section 2).

1-to-n ribbon (and vice-versa) : The regions of the surface that represent a possible change in the topology. For these branchings the Euler characteristic can be computed similar to the 1-to-1 ribbon case: close the different branches by endcaps to get a topological sphere. Hence for 1-to-n ribbons (see Fig. 3) we have $\chi = 1 - n$.

For example, in a torus there would be one 1-to-2 ribbon where the graph traversal first encounters the hole of the torus and one 2-to-1 ribbon where the hole ends. Both of these events need to be captured in order to construct the correct topology of the torus. In contrast, the surface region between these two important events is a sequence of adjacent 1-to-1 ribbons for each branch which can be discarded without changing the topology of the surface.

Since these adjacency relationships are completely determined by ribbon neighbors, ribbon construction *and* event detection can be performed in a sweep algorithm. Once the ribbons at level n are constructed, event detection is performed by walking along the previous ribbons at level $n-1$ to see if an event ribbon was encountered. For example, for each of the Surfels in ribbons at level $n-1$, we check that their descendants have the same branch number. If not, a 1-to-n ribbon has been found. Likewise by keeping track of the branch numbers already seen, a n-to-1 ribbon can be detected when different predecessor ribbons are connected to the same descendant ribbon. Finally, if a ribbon has no valid descendant ribbons, it is saved as an endcap.

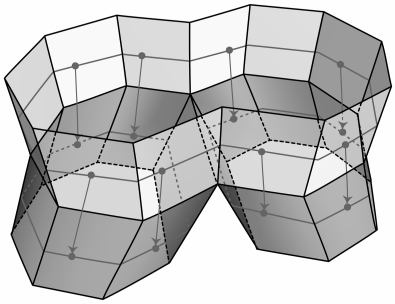


Figure 6: *1-to-n ribbon detection (n-to-1 ribbon detection is similar but inverted).*

The desired coarseness of the mesh can be controlled by adding criteria for ribbon selection. For example, consider a requirement that the initial mesh exhibit good aspect ratio triangles. This can be achieved by selecting ribbons at multiples of some integer distance w and changing the sampling density within the ribbons to also be of average distance w.

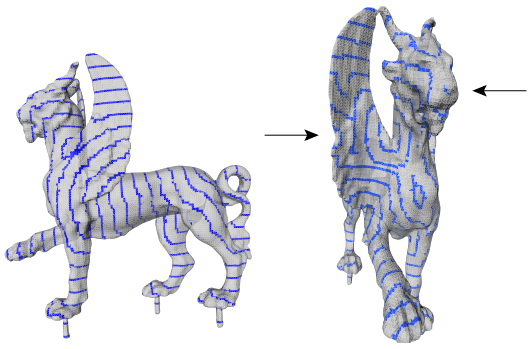


Figure 7: *On the left is the the distance ribbons for the feline dataset. The source Surfel is near the feline's tail. On the right is subsampling of the unmodified distance ribbons. There are two visible tails on the left wing and on the nose.*

Mesh Construction At this point, we have a list of all contours of the surface which are required for tiling a good coarse approximation of the final surface. The final step of our algorithm is related to contour stitching [1, 14, 12]. However, since we work within the

framework of the volume data we do not face the traditional correspondence problems of contour stitching. Specifically, the volume data and the topological graph prevent ambiguities about inter-contour connections.

Ribbon Subsampling and Shortest Distance Projection The general procedure is to subsample each ribbon along its length to convert it into a coarse contour of edges and vertices to be triangulated with adjacent contours. Adjacent contours are connected to one another by projecting ribbon samples to the next saved ribbon (see Figure 6). The projection step may result in samples being too close or too far away from one another due to changes in the geometry of the iso-surface. In this case we can adjust the number of samples to accommodate the density change by snapping close points together, or inserting a midpoint sample. The samples on both contours are enumerated in corresponding order to facilitate triangulation. Endcaps are evenly subsampled and connected to a central point.

Stitching It is easy to tile two contours that have a one-to-one correspondence in their sample enumeration. The general approach of our algorithm is to *break* the ribbons into one-to-one correspondence and then use bridges between adjacent connected ribbons to correctly model the topology of the surface. Thus 1-to-n ribbons and n-to-1 ribbons are conceptually handled by "breaking" them into n pairs of 1-to-1 ribbons with conforming bridges between appropriate segments (Fig. 8). This is done by making a pass around the larger ribbon to find if two neighboring samples have been projected from different predecessor ribbons, in which case they are stored to make the conforming bridge (Fig. 8). The following pseudo code outlines the stitching algorithm:

```
For all saved ribbons
    //process all m ribbons of distance n
    If a ribbon is not sampled
        evenly sample at intervals of w Surfels
    //else the ribbon may already be sampled from previous projection
    For each sample of the current ribbons
        Project down to next saved ribbons
    //check the spacing for the new samples
    For each Surfel of the child ribbons
        If samples too close: snap to one sample
        If samples too distant: insert a midpoint

    allocate sample lists for breaking ribbons into 1-to-1
    top-lists[m], bottom-lists[n] //n is the number of child ribbons
    //put the current and projected samples into the appropriate lists
    Traverse the current ribbon's samples
        If the current ribbon is a 1-to-n ribbon
            branch = child sample's branch number
            Put the current sample in the top-list[branch]
            Put the associated child sample in the bottom-list[branch]
        Else if the current ribbon is a n-to-1 ribbon
        //same procedure but branch = current ribbons branch number
    Triangulate the ordered samples of the corresponding top and bottom lists

    //check for edges to make conforming bridge
    If the current ribbon is a 1-to-n ribbon
    Traverse the current ribbon's samples
        If two neighbor samples have children with different branch numbers
            Store the samples until the corresponding pair is found
        Triangulate the four samples to make the conforming bridge
    Else if the current ribbon is a n-to-1 ribbon
    //same procedure but traverse the child ribbon's samples
```

It is worth noting that there is a case equivalent to a n-to-1 ribbon immediately followed by a 1-to-m ribbon. Due to the discrete nature of the samples this can appear as an *n-to-m* ribbon. This case is easily identifiable and tagged in the event detection: two child ribbons will have more than one parent in common. The previous pseudo-code applies to this special case as well.

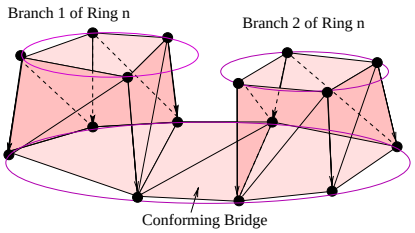


Figure 8: *Stitching example of a n-to-1 ribbon.*

2.4 Discussion

One of the benefits of this approach is the low memory overhead for the topological graph representation. In the case of an $O(n^3)$ volume the storage requirement for the distance tree is on average $O(n^2)$, as it depends on the size of the surface. The only other data that we need to store for generation of the coarse mesh is dependent on the ribbons of the topological graph which is approximately $O(n)$. Memory overhead for ribbons is minimized by keeping only, (i) the ribbons selected to be part of the coarse mesh; (ii) the last ribbon constructed and (iii) the current ribbon, which is being evaluated for possible selection. Although both our algorithm and MC use total storage of $O(n^2)$ on average, our algorithm has a more compact runtime footprint than a typical MC implementation. In particular, a time efficient implementation of the MC algorithm typically keeps information for all the voxels on the surface. This requires storage of three float values associated with each edge intersection (up to 36 floats per voxel) and three integers per face (up to 12 integers per voxel). In contrast, our algorithm does not require such detailed storage and only requires one integer and one pointer per voxel. Furthermore, we have presented the algorithm as if a distance value is permanently stored for each Surfel. This is only true conceptually, as distance values can be stored temporarily and only for voxels on the *frontier* region of the sweep. The frontier region of the sweep is the region of the surface between the last ribbon selected to be a part of the mesh and the current ribbon being evaluated. In addition, assuming that a subsequent simplification is performed on the MC mesh, typical algorithms will use at least an additional copy of the finest mesh and a sorted list of vertices, resulting in an even larger memory footprint than our entire coarse extraction routine.

3 Multi-Scale Force-based Solver

Once a coarse mesh with the correct topology is found, the next step of the algorithm consists of turning this initial mesh into a hierarchical triangulation fitting the data with suitable sampling densities and well shaped triangles. To solve for the iso-surface one may consider the signed distance function of the volume as a potential field and search for the minimum potential solution [24, 23, 22, 43, 38]. Unfortunately, this approach has a significant drawback: the trade-off between closeness to the data and the smoothness of the solution is hard to tune. In essence, smoothness of the solution and faithfulness to the desired goal surface compete with each other. Too much regularization will lead to smooth, unfit surfaces, while not enough regularization will lead to convergence difficulties. In both cases, the overall speed and accuracy is very dependent on fine tuning of parameters. This has been partially addressed by scheduling the regularization as decreasing in time [22]. Such strategies help, but still require careful tuning of parameters on a case by case basis.

The above approaches use the gradient of distance whose computation is notoriously unstable, especially in the presence of noise. For this reason we have chosen to use the distance itself. The current mesh approximation locally inflates or deflates based on the distance to the zero-contour. The direction of (local) motion of the mesh is given by its local normal, while the magnitude (and sign) of motion are determined by the distance function itself, similar to [40]. This approach, inspired by work in image processing [5],

has already been used with success in the context of active implicit surfaces [8, 51]. As a novel element we add a reparameterization technique to control triangle shapes and their variation across the surface. In this way, we obtain adaptive sampling and well shaped triangles without introducing forces which compete with the interpolation constraints. Since the meshes are refined through adaptive quadrisection we have a natural multiresolution structure which we exploit directly for an efficient multiscale solver. Our setup gives rise to a number of different force terms detailed below. External forces minimize the distance between the mesh and the zero-contour of the data. Internal forces arise from the reparameterization terms.

3.1 External Forces

We begin by considering the force acting on a single triangle before giving the actual equations for the net force on a vertex in the mesh. Following the balloon strategy, we define the force acting on a triangle T of our mesh as being along the normal of the triangle, with a sign and a magnitude depending on the surface integral of the distances d between the triangle and the actual zero-contour C:

$$F_T = \mathbf{n}_T / \mathcal{A}_T \int_{x \in T} d(x, C)\ dx$$

where $\mathbf{n}_T$ is the triangle normal and $\mathcal{A}_T$ is the area of T. The integral of the distance across the face can be computed exactly in the volume setting, since we assume that the distance varies linearly across a given voxel. In practice this is overkill and we use a much cheaper sampling criterion. Each triangle face is randomly sampled with a uniform distribution whose area density depends on the total area of the triangle. First, however, we compute the variance of the distance for a small number of uniform samples in order to short circuit unnecessary sampling. This results in quicker force computations, while preserving the quality of the approximation. Note that the minimum bound on the discretization rate is of the order of a voxel size, since everything is assumed to vary linearly within a voxel. Therefore, we use the following simple sampling strategy:

```
Temporarily quadrisect the triangle T into four small triangles t_i
For each t_i
    E[d] += d_i = DistanceAtBarycenter(t_i)
    E[d^2] += (d_i)^2
m_T = 4     //the number of samples
//calculate the variance V_T[d] of these distances
V_T[d] = E[d^2] - (E[d])^2
If V_T[d] ≥ δ
    m_T = A_T / a_vf     //a_vf = area of a voxel face
    For each m_T
    //stochastically sample the triangle with a uniform distribution
    E[d] += DistanceAtRandomSample(T)
```

The variance of a discrete set of distances is computed in the standard way $V_T[d] = E[d^2] - E[d]^2$, where E denotes the mean of its argument. A more sophisticated method, using fully adaptive sampling depending on variance, can be derived, but this simple approach has proved sufficient and has the advantage of being very efficient. The final net force on a triangle is be given by the above mean of the distances

$$\mathbf{F}_T = \mathbf{n}_T E[d].$$

The solver requires forces acting on vertices. To arrive at these we use the above sample points to compute integrals for each vertex by integrating over all incident triangles, weighting each sample point with its respective barycentric coordinate. Every sample point within a triangle contributes to the force integrals associated with its corner points as follows:

$$\begin{aligned} &1/m_T\ \mathbf{n}_T d(x_i, C)\ \phi_j(x_i) \\ &1/m_T\ \mathbf{n}_T d(x_i, C)\ \phi_k(x_i) \\ &1/m_T\ \mathbf{n}_T d(x_i, C)\ \phi_l(x_i) \end{aligned}$$

where $x_i \in T$ is the sample location; (j, k, l) are the corners of T; and the ϕ give the barycentric coordinate of x_i with respect to j, k, and l respectively. Effectively we are using piecewise linear finite elements and stochastic sampling to evaluate the associated integrals. In the implementation we simply iterates over all triangles and accumulates the integrals at each vertex.

With this scheme, faces will tend to move towards the zero-contour. If the mesh is coarser than the small details of the zero-contour, it will settle in an optimal position, smoothing the details. The finer the mesh is, the better the fit will be. As mentioned in [23], we also noticed that vertices tend to align with sharp features on the zero-contour.

3.2 Internal Forces

Internal forces are usually added as a regularizing term, to guide the minimization to a desirable local minimum. In our approach internal forces are mainly used to ensure good aspect ratios for the faces and to keep the sampling across the surface smoothly distributed. Usually, springs of zero rest length and identical stiffness are used to keep sample points from clustering locally and ensure uniform sampling [23]. Instead we define *reparameterization* forces which act similarly, but only along the local parameter plane, not in space.

Decoupling Smoothing and Reparameterization In recent work on mesh smoothing [48, 9], the Laplacian operator has been used extensively to denoise triangulated surfaces, using the approximation:

$$\mathcal{L}(x_i) = \frac{1}{m} \sum_{j \in N_1(i)} x_j - x_i,$$

where x_j are the neighbors of vertex x_i, and $m = \#N_1(i)$ is the number of these neighbors (valence). Note that this definition is equivalent to springs with zero rest length whenever the valence is constant throughout the mesh. This Laplacian of the mesh at a vertex can be broken down into two orthogonal components:

- a component normal to the surface, creating *shape smoothing*
- and a component in the tangent plane, fairing the *parameterization* of the mesh.

The normal vector to the surface can be found easily by normalizing the curvature normal vector $\mathbf{K}$ [9, 10]:

$$\mathbf{K}(x_i) = \frac{1}{2\mathcal{A}} \sum_{j \in N_1(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(x_i - x_j). \qquad (1)$$

For arbitrary connectivity meshes numerical evidence shows that no spurious drifting artifacts appear when the surface is modified only in the direction of $\mathbf{K}$ [9]. This decomposition into normal and tangential components separates motion into one component changing shape and one changing the parameterization. We are only interested in the latter.

Reparameterization as Tangential Laplacian Smoothing In our context shape smoothing would act *against* the external forces trying to fit the initial data. Thus we are only interested in the tangential motion of Laplacian smoothing in order to improve the quality of the discretization. This reparameterization force is defined as

$$\mathbf{T}(x_i) = \mathcal{L}(x_i) - (\mathcal{L}(x_i) \cdot \mathbf{n})\mathbf{n}, \qquad (2)$$

where $\mathbf{n}$ is the normalized $\mathbf{K}$ of Equ. 1. We also use the second Laplacian operator $\mathcal{L}^2$ [27, 9] to ensure a smoother variation of sampling rate over the surface, and suppress the normal component in the same way. By proceeding as described, we keep internal and external forces distinct, thus simplifying parameter choices.

3.3 Refinement Strategy

After an optimal solution has been found for a given mesh, we evaluate a refinement criterion over each triangle. Any triangle failing the criterion is quadrisected. This hierarchy is naturally maintained in a forest of quadtrees, one tree for each original coarsest level triangle. The solver is run anew after refinement.

The two criteria used to determine if a triangle should be refined are curvature and variance of distance. If the variance of the distance samples for a given triangle is too high, the surface underneath this particular triangle must have high curvature, and the triangle requires refinement. Using a user supplied threshold ϵ_V all triangles T with $V_T[d] \geq \epsilon_V$ are refined.

Additionally we also test the curvature of the current mesh to ensure good discretization in highly curved areas. If the three vertices of a triangle have too high a curvature compared to the area of the triangle, our solver refines the triangle to better adapt to the local geometry. For generality, we add a condition to deal with sharp features in the volume data: we invalidate the test on curvature if the variance of sampled distances is too small. Refinement will be avoided if we are already describing the surface adequately. Therefore, our second refinement criterion for a triangle $T = (x_i, x_j, x_k)$ can be written:

$$(|\mathbf{K}(x_i)| + |\mathbf{K}(x_j)| + |\mathbf{K}(x_k)|)\mathcal{A}_T \geq \epsilon_\kappa \text{ and } V_T[d] \geq \frac{\epsilon_V}{10}$$

where ϵ_κ, the maximum discrete curvature, is a user-defined value. The choice of $\epsilon_V/10$ seems reasonable in all our tests, but could be defined by the user if needed, depending on the prevalence of high frequency detail in the iso-surface. It is worth noting that ϵ_V can be viewed as a smoothing factor. For example if the user wants a smoothed version of the surface they can set ϵ_V to a higher number and the system will stop after reaching a solution with fewer triangles to approximate the surface.

3.4 Overall Solver Algorithm

Once forces have been computed for every vertex in the current mesh, vertex positions are updated through an explicit dynamics step:

$$x_i^{(t+\delta t)} = x_i^{(t)} + F_{x_i}\delta t$$

advancing the mesh in time until the approximation error does not decrease further. When advancing the mesh a restriction must be placed on the time step δt to satisfy the Courant condition: the velocity of change must not travel faster than the minimum detail in the system. This condition is simple to compute in our system and as $\delta t = m_e/M_f$, where m_e is the minimum edge length and M_f the maximum force. After a step is taken the refinement criteria are evaluated and quadrisection is performed as needed. Subsequently we solve again until convergence and continue this process until the user supplied error criteria are satisfied.

The behavior of the solver is controlled by the relative weightings of distance and reparameterization forces. We have found a factor of 2 in favor of the distance forces to work reliably for a wide variety of data sets. Similarly time steps of $\delta = 0.1$ and error thresholds of $\epsilon_\kappa = 15$ and $\epsilon_V = 10^{-4}$ have proven to work well without the need for tuning. To make the error criteria scale invariant we consider the object to occupy the unit cube.

4 Results

We have applied our algorithm to a variety of datasets and compared the results with MC reconstructions as "ground truth." Some of these are shown in Figure 9.

The top sequence illustrates the case of a MRI dataset (128^3) which was segmented through a level set method. Construction of the coarsest mesh (186 triangles) took .5 seconds. The intermediate

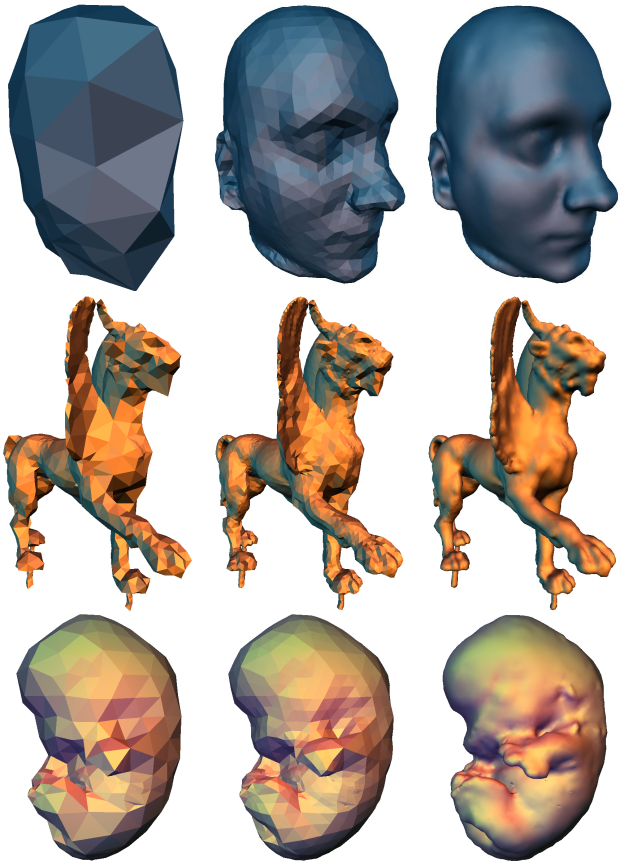


Figure 9: *Reconstructions performed with our algorithm on MRI datasets (top and bottom) and a 3D scanner generated distance function (middle). The coarsest mesh is shown on the left followed by an intermediate adaptive mesh and a final result.*

mesh contains 4810 triangles, while the final mesh has 21360 triangles. Using Metro [4] to compare our reconstruction against the MC mesh (58684 triangles) we find a relative L^2 error of $1.8*10^{-4}$ (Fig. 10). The surface is a topological sphere, but requires fairly fine levels of refinement near the ears, attesting to the performance of our solver in the presence of rapidly changing local geometric complexity.

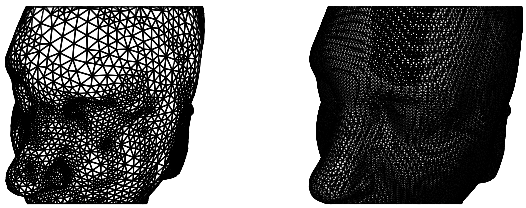


Figure 10: *Comparison between our algorithm output and a MC mesh. The relative L^2 error between these is* $1.8 * 10^{-4}$.

The middle sequence shows an extraction from a 3D scanner generated distance function [7]. The topology of the feline is non-trivial containing numerous handles in the tail region (Fig. 11) and demonstrates the performance of our coarsest level mesh extraction and topology discovery algorithm. It also demonstrates the ability of our solver to resolve fairly fine detail such as the mounting posts on the bottom of the paws. Triangle counts are 3412, 13412 and 46996 respectively (MC: 72685) for an error of $3.3*10^{-4}$. Coarsest mesh extraction time was .34 seconds on a volume of $158*74*166$ voxels.

Finally the bottom row shows another MRI dataset of a mouse

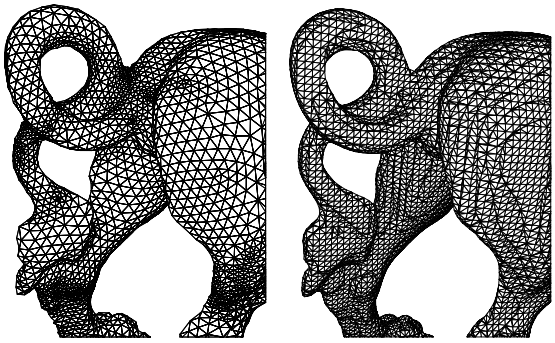


Figure 11: *Tail section of feline showing nontrivial topology. MC extraction on the right, adaptive semi-regular mesh on the left.*

embryo which was segmented with a level set method. The surface has several handles (near both front paws) and numerous concavities. All were resolved successfully. Triangle counts are 1030, 4086, and 26208 respectively (MC: 129670) with an error of $6 * 10^{-4}$. Coarsest level extraction took .78 seconds on a volume of $256 * 128^2$. Typical solver times are on the order of a few seconds for the initial meshes increasing to 4 to 5 minutes for the final reconstructions.

5 Conclusion and Future Work

We have demonstrated a novel algorithm for the capture of iso-surfaces in the form of hierarchical, adaptive semi-regular meshes. It is based on a new approach to construct a coarsest mesh with guaranteed topology approximation of the iso-surface using surface wavefront propagation to discover the topology and ensure that it is represented faithfully. In a subsequent solver step, we use a novel explicit reparameterization force employing tangential components of the first and second Laplacian of the mesh. Thus we do not have to trade off fidelity to the original data and uniqueness of the solution. The resulting meshes have a natural multiresolution structure since they are semi-regular, making them suitable for a variety of powerful digital geometry processing algorithms.

In order to avoid self-intersection problems during the solution process we have so far relied on coarsest meshes which resolve the geometry reasonably well to begin with. It would be desirable to start with the coarsest possible (in the topological sense) initial mesh and counteract any self-intersection problems in the solver itself. Other interesting areas for future work include:

- investigation of the use of multiresolution representations of the volume [16];
- optimization of the solver including adaptive time stepping strategies and automatic selection of the relative weighting for the reparameterization forces;
- application of the topological graph to irregular meshes to code topology.

Acknowledgments This work was supported in part by NSF (ACI-9624957, ACI-9721349, ACI-9982273, DMS-9874082), the NSF STC for Computer Graphics and Scientific Visualization, Alias|Wavefront, and a Packard Fellowship. A very special thanks to Eitan Grinspun, Mark Meyer, and Khrysaundt Koenig for their support and assistance. Thanks to Sean Mauch for an implementation of the Fast Marching Method, Ross Whitaker for the level set segmentation, and Ken Museth for dataset conversion. Thanks to Martin Nguyen for assistance with testing some related ideas. The head MRI data is courtesy the University of Utah's Scientific Computing and Imaging Institute, the MRI dataset of the mouse embryo courtesy the Caltech Biological Imaging Center, and the feline scan courtesy Stanford's Computer Graphics Group.

References

[1] BAJAJ, C., COYLE, E., AND LIN, K. Arbitrary Topology Shape Reconstruction from Planar Cross Sections. *Graphical Models and Image Processing 58*, 6 (1996), 524–543.

[2] BAJAJ, C., AND PASCUCCI, V. Progressive Isocontouring. Tech. Rep. TR 99-36, University of Texas at Austin, 1999.

[3] BLOOMENTHAL, J. An Implicit Surface Polygonizer. In *Graphics Gems IV*, P. S. Heckbert, Ed. Academic Press, 1994, pp. 324–349.

[4] CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum 17*, 2 (1998), 167–174.

[5] COHEN, L. D., AND COHEN, I. Finite-Element Methods for Active Contour Models and Balloons for 2D and 3D Images. *IEEE Trans. PAMI 15*, 11 (1993), 1131–1147.

[6] COHEN-OR, D., LEVIN, D., AND SOLOMIVICI, A. Three-Dimensional Distance Field Metamorphosis. *ACM Transactions on Graphics 17*, 2 (1998), 116–141.

[7] CURLESS, B., AND LEVOY, M. A Volumetric Method for Building Complex Models from Range Images. *Proceedings of SIGGRAPH 96* (1996), 303–312.

[8] DESBRUN, M., AND CANI-GASCUEL, M.-P. Active Implicit Surface for Computer Animation. In *Graphics Interface (GI'98) Proceedings*, 143–150, 1998.

[9] DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow. In *SIGGRAPH 99 Conference Proceedings*, 317–324, Aug. 1999.

[10] DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. Anisotropic Feature-Preserving Denoising of Height Fields and Bivariate Data. In *Graphics Interface'2000 Proceedings*, May 2000.

[11] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. *Proceedings of SIGGRAPH 95* (1995), 173–182.

[12] EKOULE, A. B., PEYRIN, F. C., AND ODET, C. L. A Triangulation Algorithm From Arbitrary Shaped Multiple Planar Contours. *ACM Transactions on Graphics 10*, 2 (1991), 182–199.

[13] FREEMAN, H. Computer Processing of Line-drawing Images. *ACM Computing Surveys 6*, 1 (1974), 57–97.

[14] FUCHS, H., KEDMEN, Z., AND USELTON, S. Optimal Surface Reconstruction from Planar Contours. *Communications of the ACM 20*, 10 (1977), 693–702.

[15] GARLAND, M., AND HECKBERT, P. S. Surface Simplification Using Quadric Error Metrics. *Proceedings of SIGGRAPH 96* (1996), 209–216.

[16] GERSTNER, T., AND PAJAROLA, R. Topology Preserving and Controlled Topology Simplifying Multiresolution Isosurface Extraction. *Proceedings of Visualization 00* (2000).

[17] GIBSON, S. Using Distance Maps for Accurate Surface Representation in Sampled Volumes. In *Proceedings of the 1998 Symposium on Volume Visualization*, 23–30, October 1998.

[18] GORDON, D., AND UDUPA, J. Fast Surface Tracking in Three-dimensional Binary Images. *Computer Vision, Graphics, and Image Processing 45*, 2 (Feb. 1989), 196–241.

[19] GUSKOV, I., VIDIMČE, K., SWELDENS, W., AND SCHRÖDER, P. Normal Meshes. *Proceedings of SIGGRAPH 00* (2000).

[20] HECKBERT, P. S., AND GARLAND, M. Survey of Polygonal Surface Simplification Algorithms. Tech. rep., Carnegie Mellon University, 1997.

[21] HOPPE, H. Progressive Meshes. *Proceedings of SIGGRAPH 97* (1997), 189–198.

[22] HOPPE, H., DEROSE, T., DUCHAMP, T., HALSTEAD, M., JIN, H., MCDONALD, J., SCHWEITZER, J., AND STUETZLE, W. Piecewise Smooth Surface Reconstruction. *Proceedings of SIGGRAPH 94* (1994), 295–302.

[23] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. Mesh Optimization. *Proceedings of SIGGRAPH 93* (1993), 19–26.

[24] KASS, M., WITKIN, A., AND TERZOPOULOS, D. Snakes: Active Contour Models. In *1st Conference on Computer Vision*, 321,331, June 1988.

[25] KHODAKOVSKY, A., SCHRÖDER, P., AND SWELDENS, W. Progressive Geometry Compression. *Proceedings of SIGGRAPH 00* (2000).

[26] KIMMEL, R., AND SETHIAN, J. Fast Marching Method on Triangulated Domains. In *Proceedings of the National Academy of Science*, vol. 95, 8341–8435, 1998.

[27] KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. Interactive Multi-Resolution Modeling on Arbitrary Meshes. In *SIGGRAPH 98 Conference Proceedings*, 105–114, July 1998.

[28] KOBBELT, L. P., VORSATZ, J., LABSIK, U., AND SEIDEL, H.-P. A Shrink Wrapping Approach to Remeshing Polygonal Surfaces. *Computer Graphics Forum 18*, 3 (1999), 119–130.

[29] KRISHNAMURTHY, V., AND LEVOY, M. Fitting Smooth Surfaces to Dense Polygon Meshes. *Proceedings of SIGGRAPH 96* (1996), 313–324.

[30] LACHAUD, J.-O. Topologically Defined Iso-surfaces. Research Report 96-20, Laboratoire de l'Informatique du Parallélisme, ENS Lyon, France, 1996.

[31] LACHAUD, J.-O., AND MONTANVERT, A. Deformable Meshes with Automated Topology Changes for Coarse-to-fine 3D Surface Extraction. *Medical Image Analysis 3*, 2 (1999), 187–207.

[32] LAZARUS, F., AND VERROUST, A. Level Set Diagrams of Polyhedral Objects. In *Proceedings of the Fifth Symposium on Solid Modeling and Applications*, 130–140, June 1999.

[33] LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. MAPS: Multiresolution Adaptive Parameterization of Surfaces. *Proceedings of SIGGRAPH 98* (1998), 95–104.

[34] LINDSTROM, P. Out-of-Core Simplification of Large Polygonal Models. In *Computer Graphics (SIGGRAPH '00 Proceedings)*, July 2000.

[35] LINDSTROM, P., AND TURK, G. Evaluation of Memoryless Simplification. *IEEE Transactions on Visualization and Computer Graphics 5*, 2 (1999), 98–115.

[36] LORENSEN, W., AND CLINE, H. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (Proceedings of Siggraph '87 21*, 4 (1987), 163–169.

[37] MALLADI, R., SETHIAN, J., AND VEMURI, B. Shape Modeling with Front Propagation: A Level Set Approach. *IEEE Trans. PAMI 17*, 2 (1995), 158–175.

[38] MCINERNEY, T., AND TERZOPOULOS, D. Deformable Models in Medical Image Analysis: a Survey. *Medical Image Analysis 1*, 2 (1996), 91–108.

[39] MCINERNEY, T., AND TERZOPOULOS, D. Topology Adaptive Deformable Surfaces for Medical Image Volume Segmentation. *IEEE Transactions on Medical Imaging 18*, 10 (1999), 840–850.

[40] MILLER, J. V., BREEN, D. E., LORENSEN, W. E., O'BARA, R. M., AND WOZNY, M. J. Geometrically Deformed Models: A Method for Extracting Closed Geometric Models from Volume Data. *Computer Graphics (Proceedings of SIGGRAPH 91) 25*, 4 (1991), 217–226.

[41] MOISE, E. E. *Geometric Topology in Dimensions 2 and 3*. Springer-Verlag, New York, USA, 1977.

[42] PAYNE, B., AND TOGA, A. Distance Field Manipulation of Surface Models. *IEEE Computer Graphics and Applications 12*, 1 (1992), 65–71.

[43] QIN, H., MANDAL, C., AND VEMURI, B. C. Dynamic Catmull-Clark Subdivision Surfaces. *IEEE Transactions on Visualization and Computer Graphics 4*, 3 (1998), 215–229.

[44] SETHIAN, J. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, UK, 1999.

[45] SHINAGAWA, Y., KERGOSIEN, Y. L., AND KUNII, T. L. Surface Coding Based on Morse Theory. *IEEE Computer Graphics and Applications 11*, 5 (1991), 66–78.

[46] SHINAGAWA, Y., AND KUNII, T. L. Constructing a Reeb Graph Automatically from Cross Sections. *IEEE Computer Graphics and Applications 11*, 6 (1991), 44–51.

[47] STANDER, B. T., AND HART, J. C. Guaranteeing the Topology of an Implicit Surface Polygonization for Interactive Modeling. In *Computer Graphics (SIGGRAPH '97 Proceedings)*, 279–286, August 1997.

[48] TAUBIN, G. A Signal Processing Approach to Fair Surface Design. In *SIGGRAPH 95 Conference Proceedings*, 351–358, Aug. 1995.

[49] VERROUST, A., AND LAZARUS, F. Extracting Skeletal Curves from 3D Scattered Data. *The Visual Computer 16*, 1 (2000), 15–25.

[50] WESTERMANN, R., KOBBELT, L., AND ERTL, T. Real-time Exploration of Regular Volume Data by Adaptive Reconstruction of Isosurfaces. *The Visual Computer 15*, 2 (1999), 100–111.

[51] WHITAKER, R., AND BREEN, D. Level-Set Models for the Deformation of Solid Objects. In *Proceedings of the Third International Workshop on Implicit Surfaces*, 19–35, June 1998.

[52] WHITAKER, R. T., AND CHEN, D. T. Embedded Active Surfaces for Volume Visualization. In *SPIE Medical Imaging VIII*, 340–352, 1994.

[53] WOOD, Z. J. Semi-Regular Mesh Extraction from Volumes. Master's thesis, Caltech, Pasadena, California, 2000.

[54] ZORIN, D., AND SCHRÖDER, P., Eds. *Subdivision for Modeling and Animation*. Course Notes. ACM SIGGRAPH, 1999.

[55] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interactive Multiresolution Mesh Editing. *Proceedings of SIGGRAPH 97* (1997), 259–268.

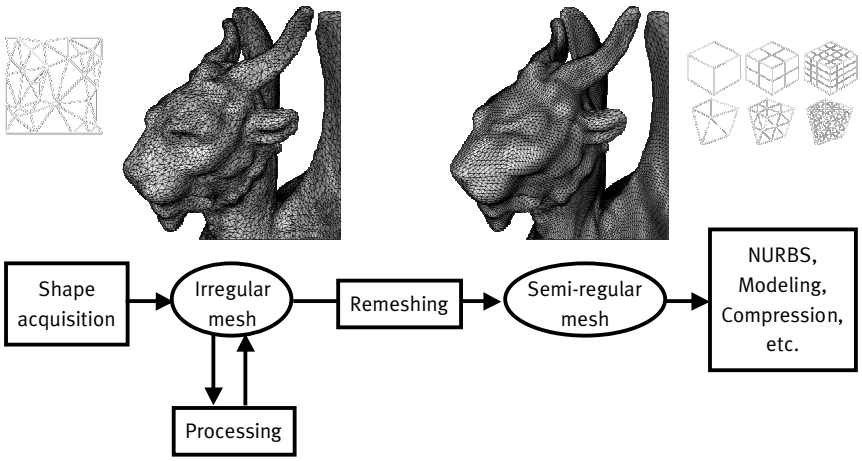
Writing a remesher

Igor Guskov
Caltech

Outline

- Introduction
- Parameterization
  - within one patch
- Patch layout techniques
  - many patches
- Normal sampling

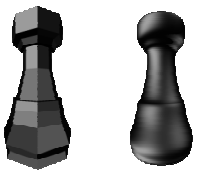
Overview

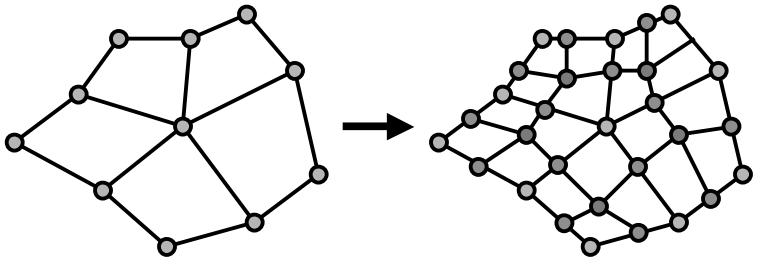


Subdivision

Mesh refines

- no new information
  - smooth surface results

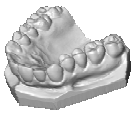


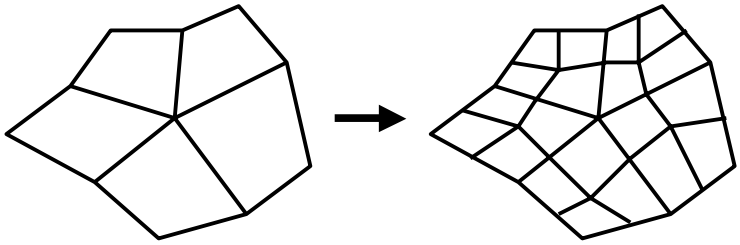


REMESHING

Mesh refines

- new information added
  - sample original geometry





REMESHING

Input

- dense irregular mesh

Output

- dense semi-regular mesh

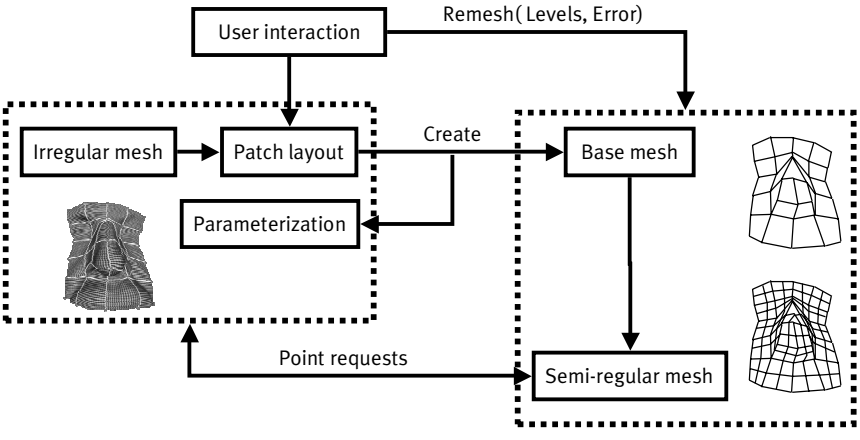
Goal

- small error

REMESHER

- Mesh is loaded
- Layout is specified
- Semi-regular mesh refines
  - like subdivision but
    - new points are on original surface
    - error estimate governs adaptivity

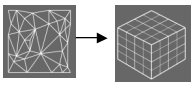
REMESHER



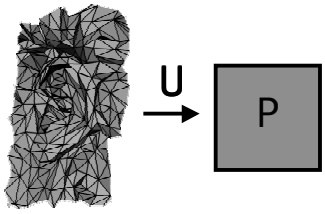
Outline

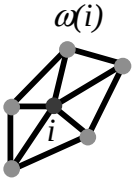
- Introduction
- Parameterization/sampling
  - within one patch
- Patch layout techniques
  - many patches
- Normal sampling

Parameterization



- Solve linear system
  - "discrete Laplacian"
  - $U[i] = \sum_j a[i,j]\, U[j]$
    - convex combination, $U = (u,v)$
  - Tutte '63 - bijection
    - planar graph drawing
  - Floater '97 - smoothness

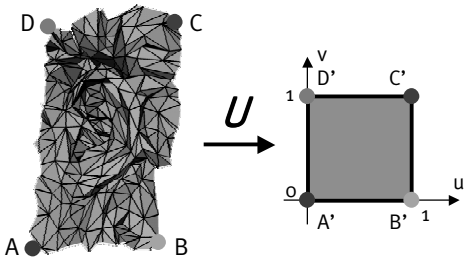




One quad patch

Parameterization

- *U*: mesh → square
  - *U(m) = (u(m),v(m))*
  - bijection
- boundary
- inside verts



Boundary

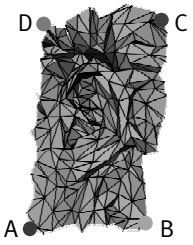
ABCDA onto A'B'C'D'A'

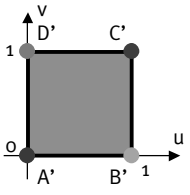
- corners

u(A)=0, u(B)=1, u(C)=1, u(D)=0

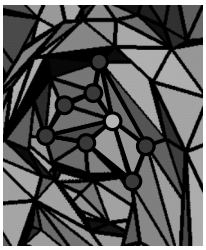
v(A)=0, v(B)=0, v(C)=1, v(D)=1

- sides
  - AB, BC, CD, DA
  - parameterize by length



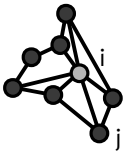


INSIDE VERTICES



Linear equation

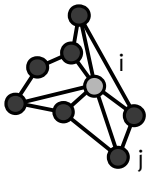
- $U[i] = \sum_{j \in ring(i)} a[i,j]\, U[j]$
  - convex combination
    - $\sum_{j \in ring(i)} a[i,j] = 1,\ a[i,j] > 0$
- Tutte '63: bijection
  - boundaries as above
  - convexity of region and coefficients



FLOATER SCHEME

Coefficients a[i,j]

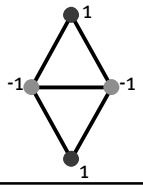
- smoothness
- Floater 1997
- discrete parameterization
  - not a discretization of some PDE
  - positive coefficients are crucial

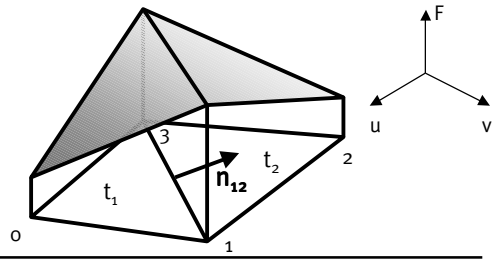


SMOOTHNESS MEASURE

Second difference

- F(u,v): scalar function
- $g(t_1,t_2) = [\text{grad } F(t_2) - \text{grad } F(t_1)] \cdot \mathbf{n}_{12}$
  $= \Sigma_i c_i F_i$

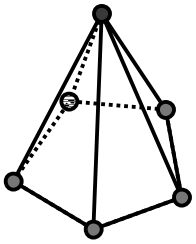


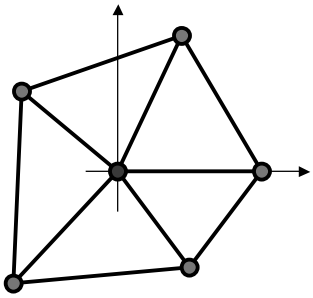


LOCAL PARAMETERS II

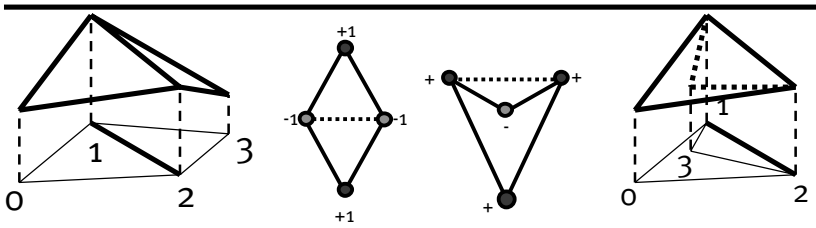
One-ring

- Normalize angles to sum to 2π
- Preserve lengths





Positive coeffs I



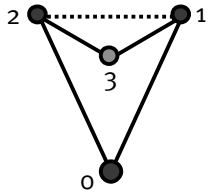
Moving pt 3 inside [0,1,2]

- coefficients $g = \Sigma_i c_i F_i$
- $c_0>0$, $c_1>0$, $c_2>0$
- $c_3<0$

Positive coeffs II

Coefficients

- $c_0 = d_{12} / A_{012}$
- $c_1 = d_{12} A_{203} / (A_{123} A_{012})$
- $c_2 = d_{12} A_{130} / (A_{123} A_{012})$
- $c_3 = - d_{12} / A_{123}$

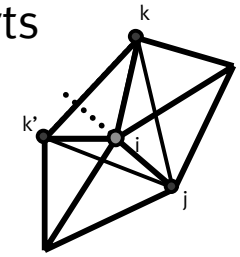


- $g(0123)[u] = c_0u_0 + c_1u_1 + c_2u_2 + c_3u_3$

Positive stencil I

Fix i. For every vertex j in one-ring(i)

- find k and k' adjacent verts "across" from A
- form g(jkk'i)[u] =: $g_j[u]$
- note that
  - $g_j[u] = c_{jj}u_j + c_{jk}u_k + c_{jk'}u_{k'} + c_{ji}u_i$

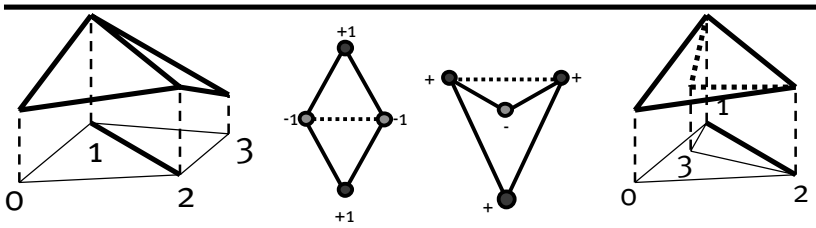


Positive stencil II

Suppose that

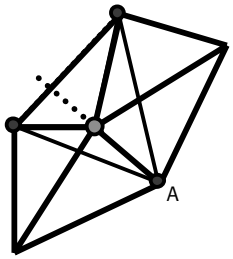
- all u_j in one-ring are known
- u_i is unknown
- solve $$\min_{u_i} \sum_{j \in \omega_1(i)} (g_j[u])^2$$
- we obtain
  - $u_i = \sum_{j \in \text{ring}(i)} a_{ij} u_j$

Positive coeffs



Floater '96

- shape-preserving
  - positive coeffs
  - smoothness



Parameterization

Assign coefficients

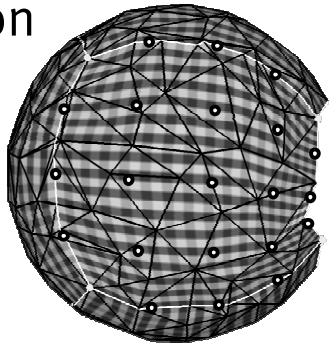
Assign boundary conditions

Form linear system

- use conjugate gradient iterative solver
- multigrid

SAMPLING

- Invert parameterization
  - find
    - triangle
    - barycentric coords



OUTLINE

- Introduction
- Parameterization
  - within one patch
- Patch layout techniques
  - many patches
- Normal sampling

PATCH LAYOUT

Manually
- click

Automatically
- MAPS: via simplification

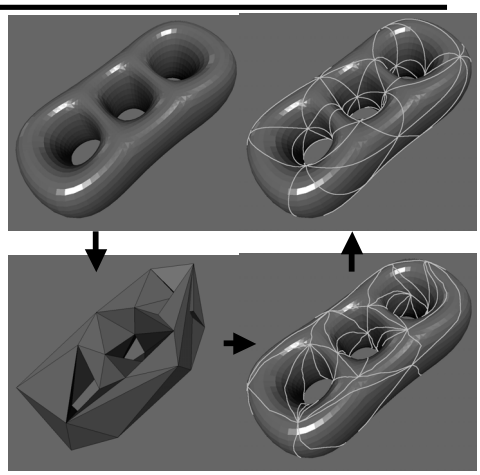
Improve layout
- relax layout (corners, curves)

AUTOMATIC LAYOUT

Base mesh
- simplification

Boundaries
- propagated

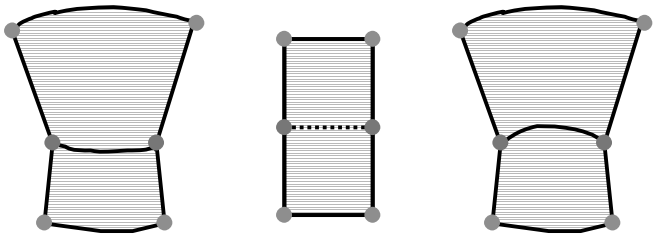
Global vertices
- relocated



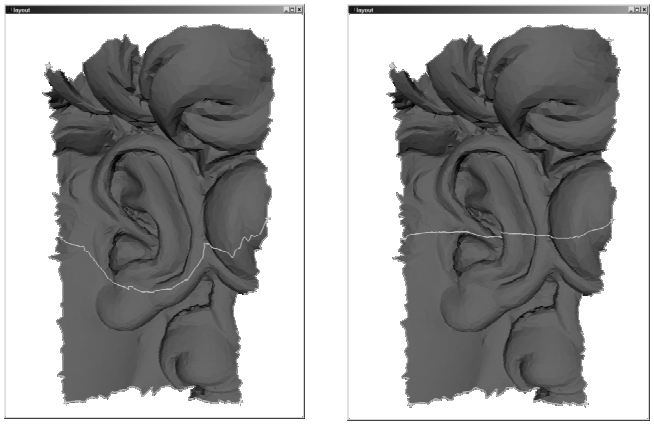
Relaxing boundaries

Pretend that we have one patch

- parameterize two patches together
- take isoline as new boundary



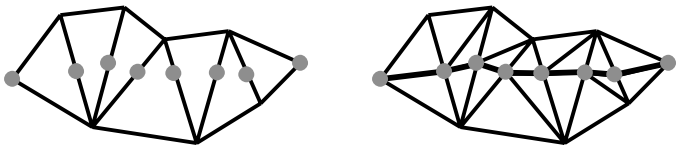
Example



Dynamic mesh

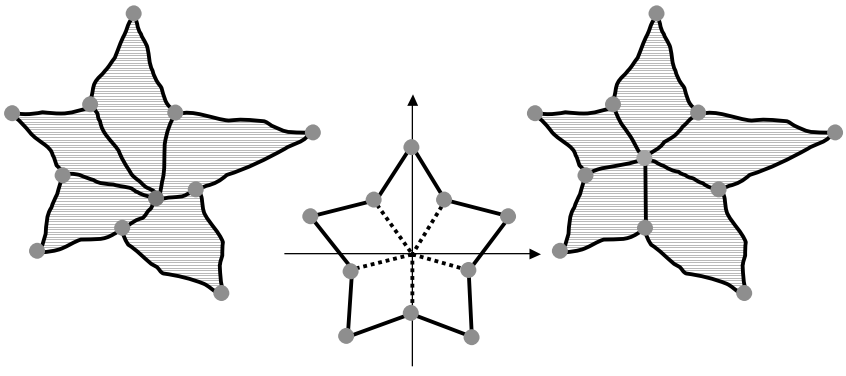
Maintain

- curves, features
- original geometry
  - bisections, edge collapses

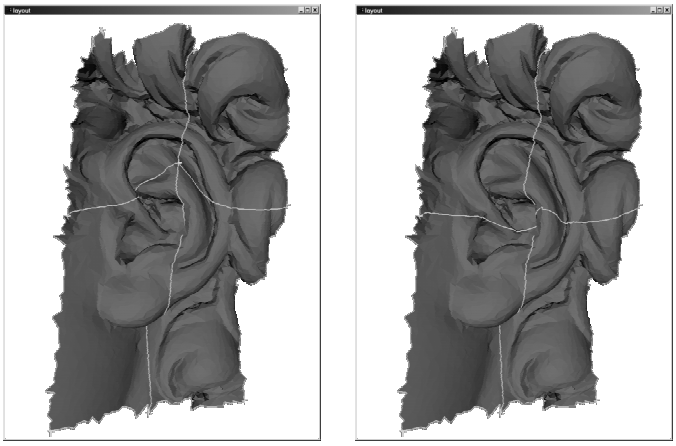


Relaxing global verts

Parameterize neighboring patches



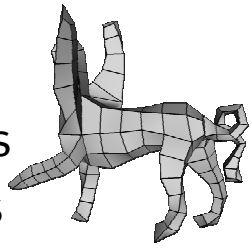
EXAMPLE



LIMB GROWING I

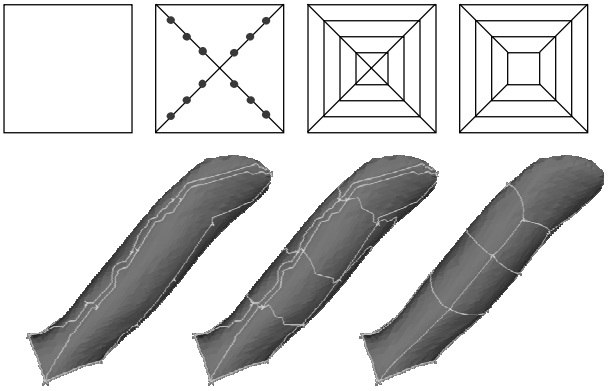
Insert several cubes

- find farthest point from boundary
- draw four geodesics
- split into x pieces
- connect like four ladders
- erase top four segments
- relax the layout



Limb growing II

Schematically



Outline

Introduction

Parameterization

- within one patch

Patch layout techniques

- many patches

Normal sampling

So much freedom

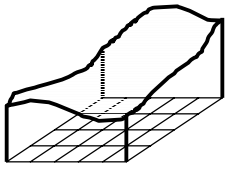
What is a good sampling?

- compact representation
  - normal meshes
    - scalar displacements at all levels
  - displaced subdivision surfaces
    - Lee et al. '2000
    - scalar displacements at one level

Normal meshes

Height fields, terrains

- f(u,v)
- one float / vertex



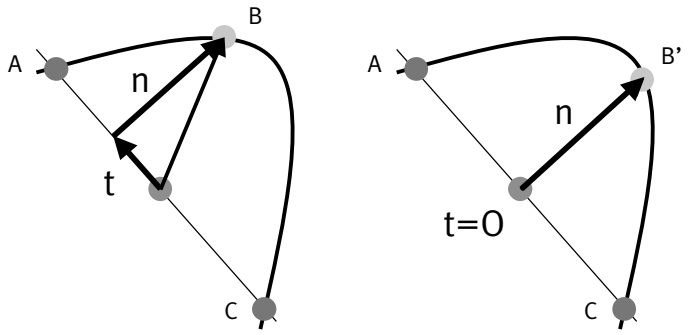
3d surfaces

- x(u,v),y(u,v),z(u,v)
- three floats / vertex

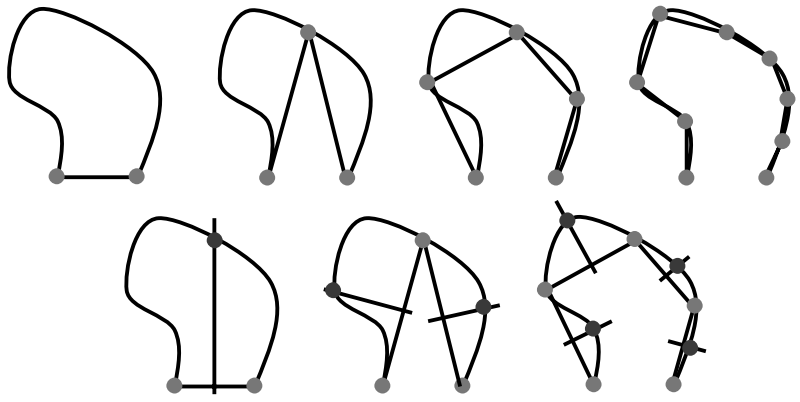
Local detail (x,y)

(t,n) stored

One float!



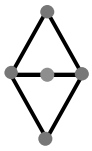
Simple algorithm

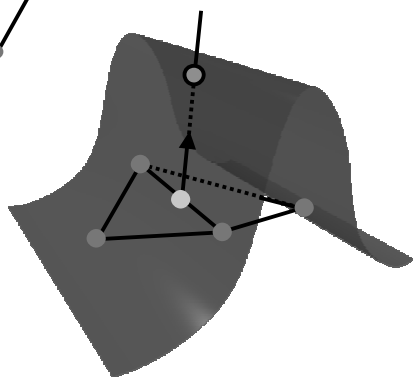


Normal Meshes

Mesh		Normal mesh
■ geometry	⋯⋯▶	same
■ connectivity	⋯⋯▶	semi-regular
■ sample locations	⋯⋯▶	optimal one float/vertex

Piercing



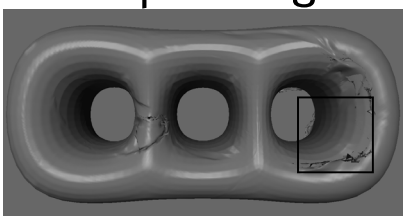


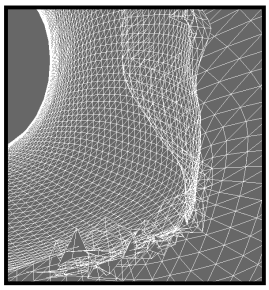
Storage
- one float/vertex

Naïve algorithm:
- start with coarse mesh
- pierce recursively

BUT...

Naïve piercing does not work!





We need

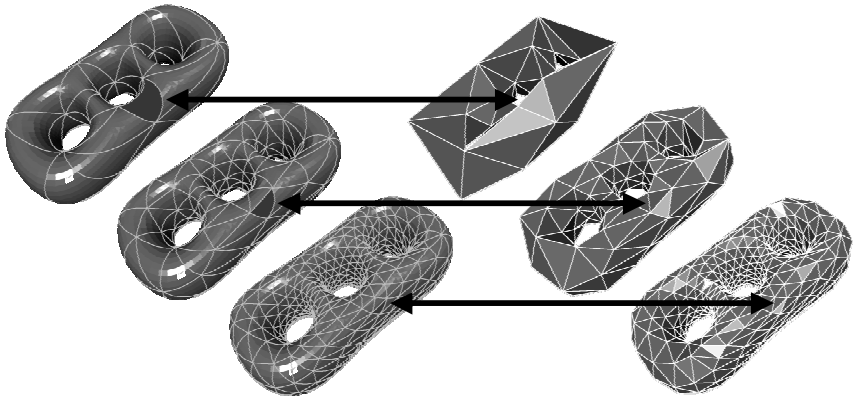
- more control over the process

ALGORITHM

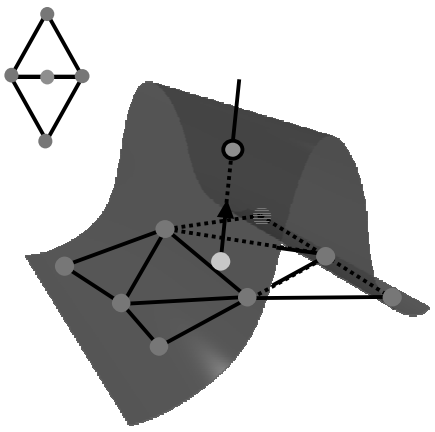
1. Initial patch layout
   - progressive hierarchy
   - global relaxation
2. Remeshing procedure
   - piercing step
   - one-to-one correspondence

Remeshing Procedure

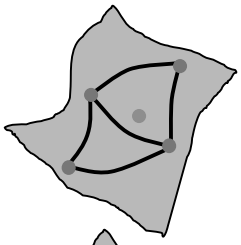
Correspondence: patches ↔ faces



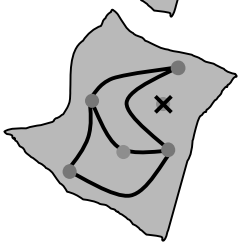
Piercing & Patches



Canonical:



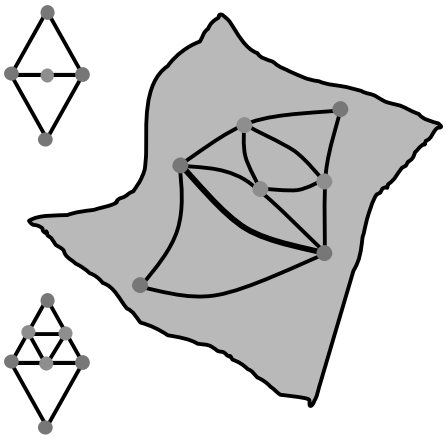
Exception:



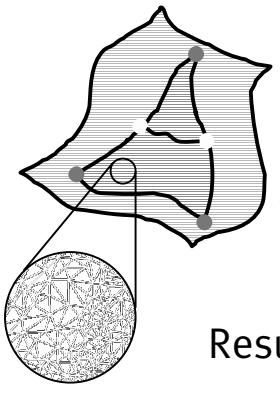
Maintenance

Operations

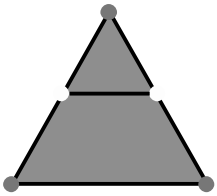
- new vertex
  - curve updated
- new edges
  - new curves



Drawing Curves



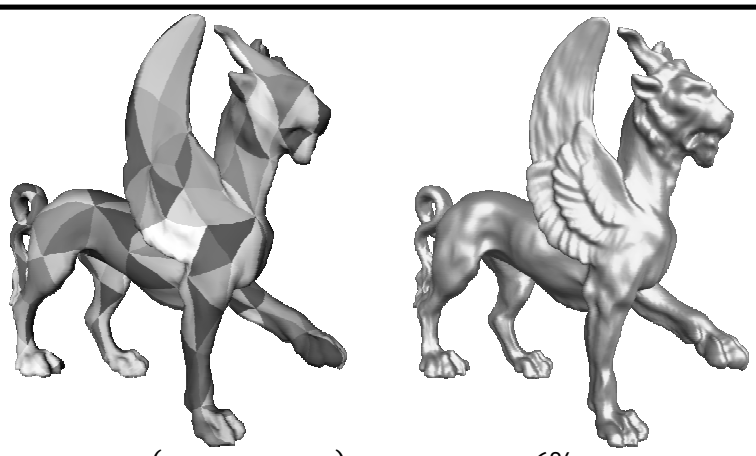
1. parameterize



2. draw in the plane

Result: curve on the original surface

Results



(mean-square) error = 0.0056%

Normal Meshes

Igor Guskov
Caltech

Kiril Vidimče
Mississippi State University

Wim Sweldens
Bell Laboratories

Peter Schröder
Caltech

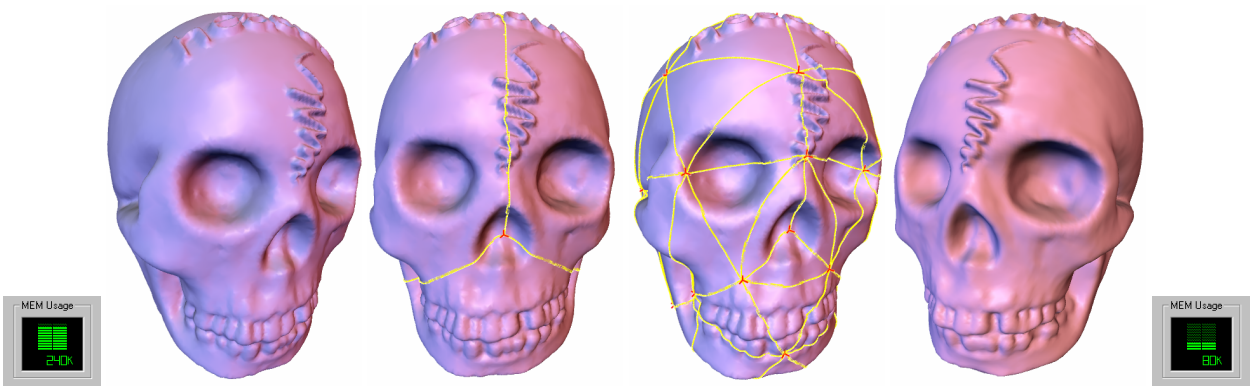


Figure 1: *Left: original mesh (3 floats/vertex). Middle: two stages of our algorithm. Right: normal mesh (1 float/vertex). (Skull dataset courtesy Headus, Inc.)*

Abstract

Normal meshes are new fundamental surface descriptions inspired by differential geometry. A normal mesh is a multiresolution mesh where each level can be written as a normal offset from a coarser version. Hence the mesh can be stored with a single float per vertex. We present an algorithm to approximate any surface arbitrarily closely with a normal semi-regular mesh. Normal meshes can be useful in numerous applications such as compression, filtering, rendering, texturing, and modeling.

CR Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling - *curve, surface, solid, and object representations; hierarchy and geometric transformations*; G.1.2 [**Numerical Analysis**]: Approximation - *approximation of surfaces and contours, wavelets and fractals*

Additional Keywords: Meshes, subdivision, irregular connectivity, surface parameterization, multiresolution, wavelets.

1 Introduction

The standard way to parameterize a surface involves *three* scalar functions $x(u,v)$, $y(u,v)$, $z(u,v)$. Yet differential geometry teaches us that smooth surfaces locally can be described by a *single* scalar height function over the tangent plane. Loosely speaking one can say that the geometric information of a surface can be contained in only a single dimension, the height over this plane. This observation holds infinitesimally; only special cases such as terrains and star-shaped surfaces can globally be described with a single function.

In practice we often approximate surfaces using a triangle mesh. While describing meshes is relatively easy, they have lost much of the structure inherent in the original surface. For example, the above observation that locally a surface can be characterized by a scalar function is not reflected in the fact that we store 3 floats per vertex. In other words, the correlation between neighboring sample locations implied by the smoothness assumption is not reflected, leading to an inherently redundant representation.

While vertex locations come as 3-dimensional quantities, the above considerations tell us that locally two of those dimensions represent parametric information and only the third captures geometric, or shape, information. For a given smooth shape one may choose different parameterizations, yet the geometry remains the same. In the case of a mesh we can observe this by noticing that infinitesimal tangential motion of a vertex does not change the geometry, only the sampling pattern, or parameterization. Moving in the normal direction on the other hand changes the geometry and leaves parameter information undisturbed.

1.1 Goals and Contributions

Based on the above observations, the aim of the present paper is to compute mesh representations that only require a single scalar per vertex. We call such representations *normal meshes*. The main insight is that this can be done using multiresolution and local frames. A normal mesh has a hierarchical representation so that all detail coefficients when expressed in local frames are scalar, i.e., they only have a normal component. In the context of compression, for example, this implies that parameter information can be perfectly predicted and residual error is entirely constrained to the normal direction, i.e., contains only geometric information. Note that because of the local frames normal mesh representations are non-linear.

Of course we cannot expect a given arbitrary input mesh to possess a hierarchical representation which is normal. Instead we de-

scribe an algorithm which takes an arbitrary topology input mesh and produces a semi-regular normal mesh describing the same geometry. Aside from a small amount of base domain information, *our normal mesh transform converts an arbitrary mesh from a 3 parameter representation into a purely scalar representation*. We demonstrate our algorithm by applying it to a number of models and experimentally characterize some of the properties which make normal meshes so attractive for computations.

The study of normal meshes is of interest for a number of reasons: they

- bring our computational representations back towards the "first principles" of differential geometry;
- are very storage and bandwidth efficient, describing a surface as a succinctly specified base shape plus a hierarchical normal map;
- are an excellent representation for compression since all variance is "squeezed" into a single dimension.

1.2 Related Work

Efficient representations for irregular connectivity meshes have been pursued by a number of researchers. This research is motivated by our ability to acquire densely sampled, highly detailed scans of real world objects [19] and the need to manipulate these efficiently. Semi-regular—or subdivision connectivity—meshes offer many advantages over the irregular setting due of their well developed mathematical foundations and data structure simplicity [23]; many powerful algorithms require their input to be in semi-regular form [21, 22, 25, 1]. This has led to the development of a number of algorithms to convert existing irregular meshes to semi-regular form through remeshing. Eck et al. [9] use Voronoi tiling and harmonic maps to build a parameterization and remesh onto a semi-regular mesh. Krischnamurthy and Levoy [15] demonstrated user driven remeshing for the case of bi-cubic patches, while Lee et al. [18] proposed an algorithm based on feature driven mesh reduction to develop smooth parameterizations of meshes in an automatic fashion. These methods use the parameterization subsequently for semi-regular remeshing.

Our work is related to these approaches in that we also construct a semi-regular mesh from an arbitrary connectivity input mesh. However, in previous work prediction residuals, or detail vectors, were not optimized to have properties such as normality. The main focus was on the establishment of a smooth parameterization which was then semi-regularly sampled.

The discussion of parameter versus geometry information originates in the work done on irregular curve and surface subdivision [4] [13] and intrinsic curvature normal flow [5]. There it is shown that unless one has the correct parameter side information, it is not possible to build an irregular smooth subdivision scheme. While such schemes are useful for editing and texturing applications, they cannot be used for succinct representations because the parameter side-information needed is excessive. In the case of normal meshes these issues are entirely circumvented in that all parameter information vanishes and the mesh is reduced to purely geometric, i.e., scalar in the normal direction, information.

Finally, we mention the connection to displacement maps [3], and in particular normal displacement maps. These are popular for modeling purposes and used extensively in high end rendering systems such as RenderMan. In a sense we are solving here the associated inverse problem. Given some geometry, find a simpler geometry and a set of normal displacements which together are equivalent to the original geometry. Typically, normal displacement maps are single level, whereas we aim to build them in a fully hierarchical way. For example, single level displacements maps were used in [15] to capture the fine detail of a 3D photography model. Cohen et al. [2] sampled normal fields of geometry and maintained these in texture maps during simplification. While these approaches all differ significantly from our interests here, it is clear that maps of this and related nature are of great interest in many contexts.

In independent work, Lee et al. pursue a goal similar to ours [17]. They introduce displaced subdivision surfaces which can be seen as a two level normal mesh. Because only two levels are used, the base domain typically contains more triangles than in our case. Also the normal offsets are oversampled while in our case, the normal offsets are critically sampled.

2 Normal Polylines

Before we look at surfaces and normal meshes, we introduce some of the concepts using curves and normal polylines. A curve in the plane is described by a pair of parametric functions $\mathbf{s}(t) = (x(t), y(t))$ with $t \in [0, 1]$. We would like to describe the points on the curve with a single scalar function. In practice one uses polylines to approximate the function. Let $\mathbf{l}(\mathbf{p}, \mathbf{p}')$ be the linear segment between the points $\mathbf{p}$ and $\mathbf{p}'$. A standard way to build a polyline multiresolution approximation is to sample the curve at points $\mathbf{s}_{j,k}$ where $\mathbf{s}_{j,k} = \mathbf{s}_{j+1,2k}$ and define the jth level approximation as

$$\mathbf{L}_j = \bigcup_{0 \le k < 2^j} \mathbf{l}(\mathbf{s}_{j,k}, \mathbf{s}_{j,k+1}).$$

To move from $\mathbf{L}_j$ to $\mathbf{L}_{j+1}$ we need to insert the points $\mathbf{s}_{j+1,2k+1}$ (Figure 2, left). Clearly this requires two scalars: the two coordinates of $\mathbf{s}_{j+1,2k+1}$. Alternatively one could compute the difference $\mathbf{s}_{j+1,2k+1} - \mathbf{m}$ between the new point and some predicted point $\mathbf{m}$, say the midpoint of the neighboring points $\mathbf{s}_{j,k}$ and $\mathbf{s}_{j,k+1}$. This detail has a tangential component $\mathbf{m} - \mathbf{b}$ and a normal component $\mathbf{b} - \mathbf{s}_{j+1,2k+1}$. The normal component is the *geometric* information while the tangential component is the *parameter* information. The way to build polylines that can be described with one

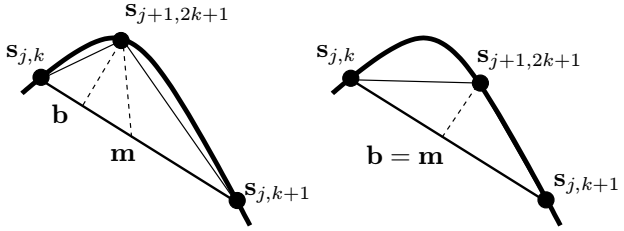


Figure 2: *Removing one point $\mathbf{s}_{j+1,2k+1}$ in a polyline multiresolution and recording the difference with the midpoint $\mathbf{m}$. On the left a general polyline where the detail has both a normal and a tangential component. On the right a normal polyline where the detail is purely normal.*

scalar per point, is to make sure that the parameter information is always zero, i.e., $\mathbf{b} = \mathbf{m}$, see Figure 2, right. If the triangle $\mathbf{s}_{j,k}$, $\mathbf{s}_{j+1,2k+1}$, $\mathbf{s}_{j,k+1}$ is Isosceles, there is no parameter information. Consequently we say that a polyline is normal if a multiresolution structure exists where every removed point forms an Isosceles triangle with its neighbors. Then there is zero parameter information and the polyline can be represented with one scalar per point, namely the normal component of the associated detail.

For a general polyline the removed triangles are hardly ever exactly Isosceles and hence the polyline is not normal. Below we describe a procedure to build a normal polyline approximation for any continuous curve. The easiest is to start building Isosceles triangles from the coarsest level. Start with the first base $\mathbf{l}(\mathbf{s}_{0,0}, \mathbf{s}_{0,1})$, see Figure 3. Next take its midpoint and check where the normal direction crosses the curve. Because the curve is continuous, there has to be at least one such point. If there are multiple pick any one.

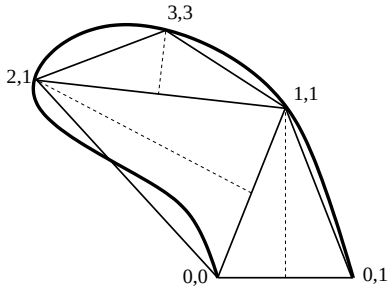


Figure 3: *Construction of a normal polyline. We start with the coarsest level and each time check where the normal to the midpoint crosses the curve. For simplicity only the indices of the $\mathbf{s}_{j,k}$ points are shown and only certain segments are subdivided. The polyline $(0,0)-(2,1)-(3,3)-(1,1)-(0,1)$ is determined by its endpoints and three scalars, the heights of the Isosceles triangles.*

Call this point $\mathbf{s}_{1,1}$ and define the first triangle. Now split the curve into two parts and repeat the procedure on each subcurve. Each time $\mathbf{s}_{j+1,2k+1}$ is found where the normal to the midpoint of $\mathbf{s}_{j,k}$ and $\mathbf{s}_{j,k+1}$ crosses the portion of the curve between $\mathbf{s}_{j,k}$ and $\mathbf{s}_{j,k+1}$. Thus any continuous curve can be approximated arbitrarily closely with a normal polyline. The result is a series of polylines $\mathbf{L}_j$ all of which are normal with respect to midpoint prediction. Effectively each level is parameterized with respect to the one coarser level. Because the polylines are normal, only a single scalar value, the normal component, needs to be recorded for each point. We have a polyline with no parameter information.

One can also consider normal polylines with respect to fancier predictors. For example one could compute a base point and normal estimate using the well known 4 point rule. Essentially any predictor which only depends on the coarser level is allowed. For example one can also use irregular schemes [4]. Also one does not need to follow the standard way of building levels by downsampling every other point, but instead could take any ordering. This leads to the following definition of a normal polyline:

Definition 1 *A polyline is normal if a removal order of the points exists such that each removed point lies in the normal direction from a base point, where the normal direction and base point only depend on the remaining points.*

Hence a normal polyline is completely determined by a scalar component per vertex.

Normal polylines are closely related to certain well known fractal curves such as the Koch Snowflake[1], see Figure 4. Here each time a line segment is divided into three subsegments. The left and right get a normal coefficient of zero, while the middle receives a normal coefficient such that the resulting triangle is equilateral. Hence the polylines leading to the snowflake are normal with respect to midpoint subdivision.

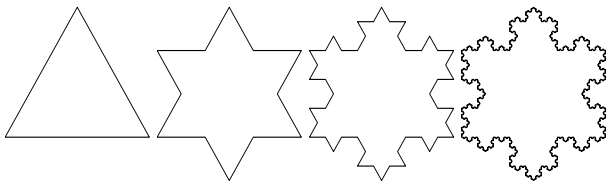


Figure 4: *Four normal polylines converging to the Koch snowflake.*

[1]Niels Fabian Helge von Koch (Sweden, 1870-1924)

There is also a close connection with wavelets. The normal coefficients can be seen as a piecewise linear wavelet transform of the original curve. Because the tangential components are always zero there are half as many wavelet coefficients as there are original scalar coefficients. Thus one saves 50% memory right away. In addition of course the wavelets have their usual decorrelation properties. In the functional case the above transform corresponds to an unlifted interpolating piecewise linear wavelet transform as introduced by Donoho [6]. There it is shown that interpolating wavelets with no primal, but many dual moments are well suited for smooth functions. Unlike in the function setting, not all wavelets from the same level j have the same physical scale. Here the scale of each coefficient is essentially the length of the base of its Isosecles triangle.

3 Normal Meshes

We begin by establishing terminology. A triangle mesh $\mathcal{M}$ is a pair $(\mathcal{P}, \mathcal{K})$, where $\mathcal{P}$ is a set of N point positions $\mathcal{P} = \{\mathbf{p}_i = (x_i, y_i, z_i) \in \mathbf{R}^3 \mid 1 \leq i \leq N\}$, and $\mathcal{K}$ is an *abstract simplicial complex* which contains all the topological, i.e., adjacency information. The complex $\mathcal{K}$ is a set of subsets of $\{1, \ldots, N\}$. These subsets come in three types: vertices $\{i\}$, edges $\{i, j\}$, and faces $\{i, j, k\}$. Two vertices i and j are *neighbors* if $\{i, j\} \in \mathcal{E}$. The 1-ring neighbors of a vertex i form a set $\mathcal{V}(i) = \{j \mid \{i, j\} \in \mathcal{E}\}$.

We can derive a definition of normal triangle meshes inspired by the curve case. Consider a hierarchy of triangle meshes $\mathcal{M}_j$ built using mesh simplification with vertex removals. These meshes are nested in the sense that $\mathcal{P}_j \subset \mathcal{P}_{j+1}$. Take a removed vertex $\mathbf{p}_i \in \mathcal{P}_{j+1} \setminus \mathcal{P}_j$. For the mesh to be normal we need to be able to find a base point $\mathbf{b}$ and normal direction N that only depend on $\mathcal{P}_j$, so that $\mathbf{p}_i - \mathbf{b}$ lies in the direction N. This leads to the following definition.

Definition 2 *A mesh $\mathcal{M}$ is normal in case a sequence of vertex removals exists so that each removed vertex lies on a line defined by a base point and normal direction which only depends on the remaining vertices.*

Thus a normal mesh can be described by a small base domain and one scalar coefficient per vertex.

As in the curve case, a mesh is in general not normal. The chance that the difference between a removed point and a predicted base point lies exactly in a direction that only depends on the remaining vertices is essentially zero. Hence the only way to obtain a normal mesh is to change the triangulation. We decide to use semi-regular meshes, i.e., meshes whose connectivity is formed by successive quadrisection of coarse base domain faces.

As in the curve setting, the way to build a normal mesh is to start from the coarse level or base domain. For each new vertex we compute a base point as well as a normal direction and check where the line defined by the base point and normal intersects the surface. The situation, however, is much more complex than in the curve case for two reasons: (1) There could be no intersection point. (2) There could be many intersection points, but only one correct one.

In case there are no intersection points, strictly speaking no fully normal mesh can be built from this base domain. If that happens, we relax the definition of normal meshes some and allow a small number of cases where the new points do not lie in the normal direction. Thus the algorithm needs to find a suitable non-normal location for the new point. In case there are many intersection points the algorithm needs to figure out which one is the right one. If the wrong one is chosen the normal mesh will start folding over itself or leave creases. Any algorithm which blindly picks an intersection point is doomed.

Parameterization In order to find the right piercing point or suggest a good alternate, one needs to be able to easily navigate around the surface. The way to do this is to build a smooth parameterization of the surface region of interest. This is a basic building block of our algorithm. Several parameterization methods have been proposed and our method takes components from each of them: mesh simplification and polar maps from MAPS [18], patchwise relaxation from [9], and a specific smoothness functional similar to the one used in [10] and [20]. The algorithm will use local parameterizations which need to be computed fast and robustly. Most of them are temporary and are quickly discarded unless they can be used as a starting guess for another parameterization.

Consider a region $\mathcal{R}$ of the mesh homeomorphic to a disc that we want to parameterize onto a convex planar region $\mathcal{B}$, i.e., find a bijective map $u : \mathcal{R} \to \mathcal{B}$. The map u is fixed by a boundary condition $\partial\mathcal{R} \to \partial\mathcal{B}$ and minimizes a certain energy functional. Several functionals can be used leading to, e.g., conformal or harmonic mappings. We take an approach based on the work of Floater [10]. In short, the function u needs to satisfy the following equation in the interior:

$$u(\mathbf{p}_i) = \sum_{k \in \mathcal{V}(i)} \alpha_{ik} u(\mathbf{p}_k), \qquad (1)$$

where $\mathcal{V}(i)$ is the 1-ring neighborhood of the vertex i and the weights α_{ik} come from the shape-preserving parameterization scheme [10]. The main advantage of the Floater weights is that they are always positive, which, combined with the convexity of the parametric region, guarantees that no triangle flipping can occur within the parametric domain. This is crucial for our algorithm. Note that this is not true in general for harmonic maps which can have negative weights. We use the iterative biconjugate gradient method [12] to obtain the solution to the system (1). Given that we often have a good starting guess this converges quickly.

Algorithm Our algorithm consists of 7 stages which are described below, some of which are shown for the molecule model in Figure 5. The molecule is a highly detailed and curved model. Any naive procedure for finding normal meshes is very unlikely to succeed.

The first four stages of the algorithm prepare the ground for the piercing procedure and build the net of curves splitting the original mesh into triangular patches that are in one-to-one correspondence with the faces of the base mesh, i.e., the coarsest level of the semi-regular mesh we build.

1. Mesh simplification: We use the Garland-Heckbert [11] simplification based on half-edge collapses to create a mesh hierarchy $(\mathcal{P}_j, \mathcal{K}_j)$. We use the coarsest level $(\mathcal{P}_0, \mathcal{K}_0)$ as an initial guess for our base domain $(\mathcal{Q}_0, \mathcal{K}_0)$. The first image of Figure 5 shows the base domain for the molecule.

2. Building an initial net of curves: The purpose of this step is to connect the vertices of the base domain with a net of non intersecting curves on the different levels of the mesh simplification hierarchy. This can easily be done using the MAPS parameterization [18]. MAPS uses polar maps to build a bijection between a 1-ring and its retriangulation after the center vertex is removed. The concatenation of these maps is a bijective mapping between different levels $(\mathcal{P}_j, \mathcal{K}_j)$ in the hierarchy. The desired curves are simply the image of the base domain edges under this mapping. Because of the bijection no intersection can occur. Note that the curves start and finish at a vertex of the base domain, but need not follow the edges of the finer triangulation, i.e., they can cut across triangles. These curves define a network of triangular shaped patches corresponding to the base domain triangles. Later we will adjust these curves on some intermediate level and again use MAPS to propagate these changes to other levels. The top middle image of Figure 5 shows these curves for some intermediate level of the hierarchy.

3. Fixing the global vertices: A normal mesh is almost completely determined by the base domain. One has to choose the base domain vertices $\mathcal{Q}_0$ very carefully to reduce the number of non-normal vertices to a minimum. The coarsest level of the mesh simplification $\mathcal{P}_0$ is only a first guess. In this section we describe a procedure for repositioning the global vertices $\mathbf{q}_i$ with $\{i\} \in \mathcal{K}_0$. We impose the constraint that the $\mathbf{q}_i$ needs to coincide with some vertex $\mathbf{p}_k$ of the original mesh, but not necessarily $\mathbf{p}_i$.

The repositioning is typically done on some intermediate level j. Take a base domain vertex $\mathbf{q}_i$. We build a parameterization from the patches incident to vertex $\mathbf{q}_i$ to a disk in the plane, see Figure 6. Boundary conditions are assigned using arclength parameterization, and parameter coordinates are iteratively computed for each level j vertex inside the shaded region. It is now easy to replace the point $\mathbf{q}_i$ with any level point from $\mathcal{P}_j$ in the shaded region. In particular we let the new $\mathbf{q}_i'$ be the point of $\mathcal{P}_j$ that in the parameter domain is closest to the center of the disk. The exact center of the disk, in general, does not correspond to a vertex of the mesh.

Once a new position $\mathbf{q}_i'$ is chosen, the curves can be redrawn by taking the inverse mapping of straight lines from the new point in the parameter plane. One can keep iterating this procedure, but we found that if suffices to cycle once through all base domain vertices.

We also provide for a user controlled repositioning. Then the user can replace the center vertex with any $\mathcal{P}_j$ point in the shaded region. The algorithm again uses the parameterization to recompute the curves from that point.

The top right of Figure 5 shows the repositioned vertices. Notice how some of them like the rightmost one have moved considerably.

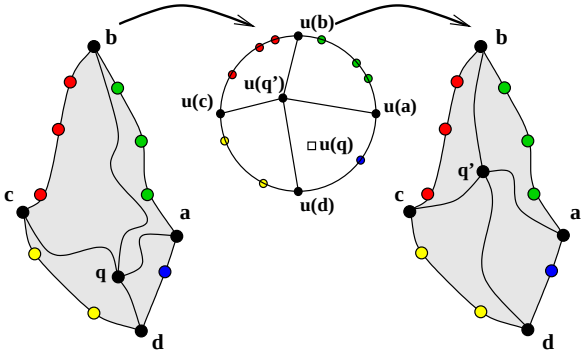


Figure 6: *Base domain vertex repositioning. Left: original patches around $\mathbf{q}_i$, middle: parameter domain, right: repositioned $\mathbf{q}_i$ and new patch boundaries. This is replaced with the vertex whose parameter coordinate are the closest to the center. The inverse mapping (right) is used to find the new position $\mathbf{q}_i'$ and the new curves.*

4. Fixing the global edges: The image of the global edges on the finest level will later be the patch boundaries of the normal mesh. For this reason we need to improve the smoothness of the associated curves at the finest level. We use a procedure similar to [9]. For each base domain edge $\{i, k\}$ we consider the region formed on the finest level mesh by its two incident patches. Let l and m be the opposing global vertices. We then compute a parameter function ρ within the diamond-shaped region of the surface. The boundary condition is set as $\rho(\mathbf{q}_i) = \rho(\mathbf{q}_k) = 0$, $\rho(\mathbf{q}_l) = 1$, $\rho(\mathbf{q}_m) = -1$, with linear variation along the edges. We then compute the parameterization and let its zero level set be our new curve. Again one could iterate this procedure till convergence but in practice one cycle suffices. The curves of the top right image in Figure 5 are the result of the curve smoothing on the finest level.

Note that a similar result can be achieved by allowing the user to position the global vertices and draw the boundaries of the patches manually. Indeed, the following steps of the algorithm do not depend on how the initial net of surface curves is produced.

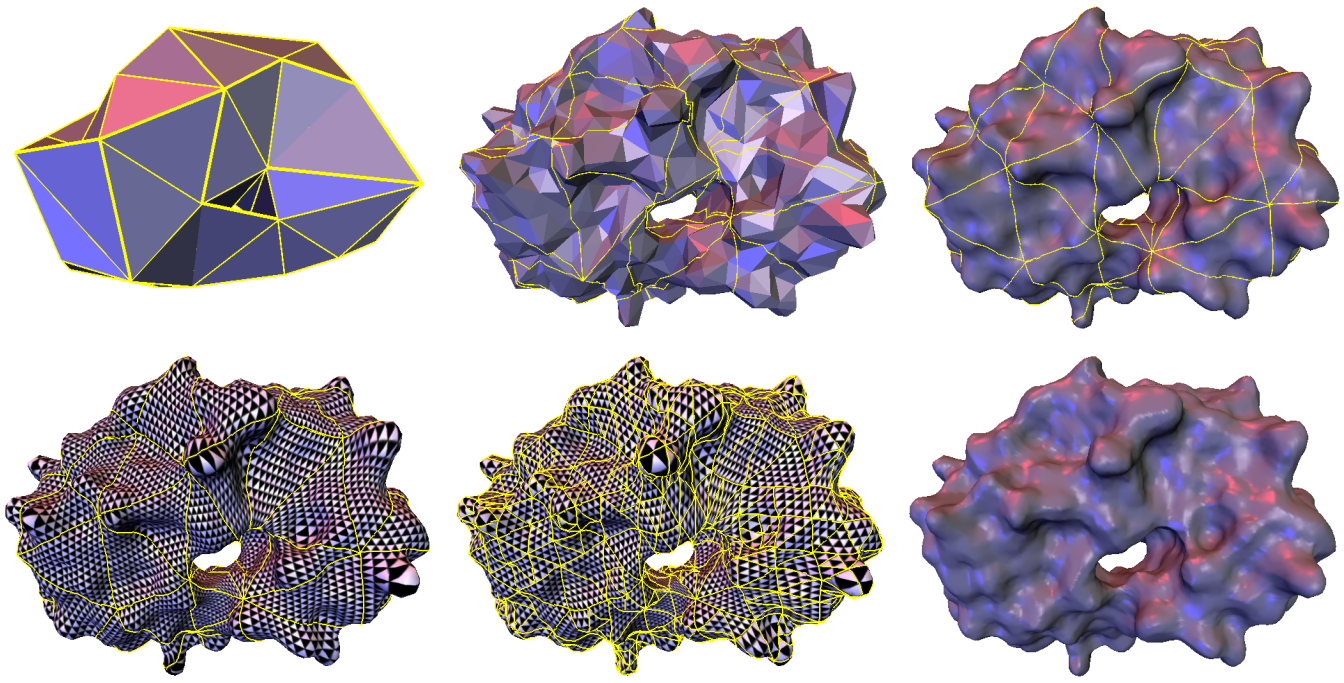


Figure 5: *The entire procedure shown for the molecule model. 1. Base domain. 2. Initial set of curves. 3. Global vertex repositioning 4. Initial Parameterization 5. Adjusting parameterization 6. Final normal mesh. (HIV protease surface model courtesy of Arthur Olson, The Scripps Research Institute)*

5. Initial parameterization: Once the global vertices and edges are fixed, one can start filling in the interior. This is done by computing the parameterization of each patch to a triangle while keeping the boundary fixed. The parameter coordinates from the last stage can serve as a good initial guess. We now have a smooth global parameterization. This parameterization is shown in the bottom left of Figure 5. Each triangle is given a triangular checkerboard texture to illustrate the parameterization.

6. Piercing: In this stage of the algorithm we start building the actual normal mesh. The canonical step is for a new vertex of the semi-regular mesh to find its position on the original mesh. In quadrisection every edge of level j generates a new vertex on level $j+1$. We first compute a base point using interpolating Butterfly subdivision [8] [24] as well as an approximation of the normal. This defines a straight line. This line may have multiple intersection points in which case we need to find the right one, or it could have none, in which case we need to come up with a good alternate.

Suppose that we need to produce the new vertex $\mathbf{q}$ that lies halfway along the edge $\{\mathbf{a}, \mathbf{c}\}$ with incident triangles $\{\mathbf{a}, \mathbf{c}, \mathbf{b}\}$ and $\{\mathbf{c}, \mathbf{a}, \mathbf{d}\}$, see Figure 7. Let the two incident patches form the region $\mathcal{R}$.

Build the straight line L defined by the base point $\mathbf{s}$ predicted by the Butterfly subdivision rule [24] and the direction of the normal computed from the coarser level points. We find all the intersection points of L with the region $\mathcal{R}$ by checking all triangles inside.

If there is no intersection we take the point $\mathbf{v}$ that lies midway between the points $\mathbf{a}$ and $\mathbf{c}$ in the parameter domain: $u(\mathbf{v}) = (u(\mathbf{a}) + u(\mathbf{c}))/2$. This is the same point a standard parameterization based remesher would use. Note that in this case the detail vector is non-normal and its three components need to be stored.

In the case when there exist several intersections of the mesh region $\mathcal{R}$ with the piercing line L we choose the intersection point that is closest to the point $u(\mathbf{v})$ in the parameter domain. Let us denote by $u(\mathbf{q})$ the parametric coordinates of that piercing point.

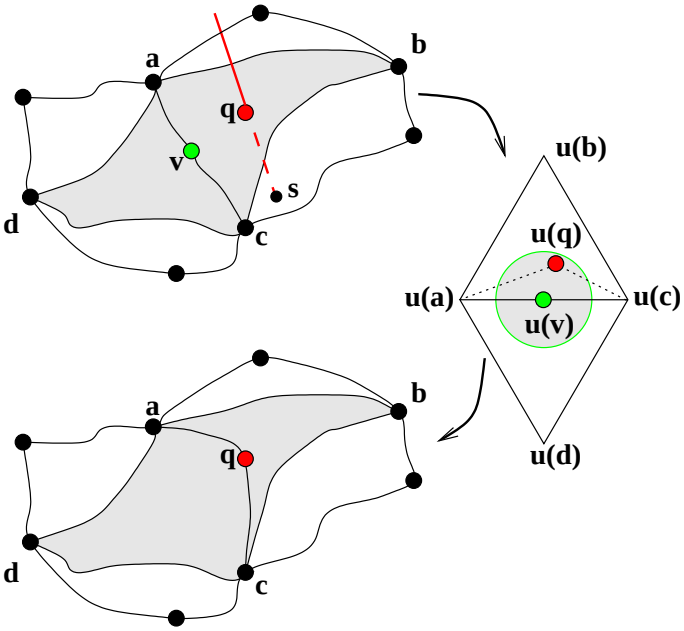


Figure 7: *Upper left: piercing, the Butterfly point is* $\mathbf{s}$*, the surface is pierced at the point* $\mathbf{q}$*, the parametrically suggested point* $\mathbf{v}$ *lies on the curve separating two regions of the mesh. Right: parameter domain, the pierced point falls inside the aperture and gets accepted. Lower left: the parameterization is adjusted to let the curve pass through* q*.*

We accept this point as a valid point of the semi-regular mesh if $\|u(\mathbf{q}) - u(\mathbf{v})\| < \kappa \|u(\mathbf{a}) - u(\mathbf{v})\|$, where κ is an "aperture" parameter that specifies how much the parameter value of a pierced

point is allowed to deviate from the center of the diamond. Otherwise, the piercing point is rejected and the mesh takes the point with the parameter value $u(\mathbf{v})$, resulting in a non-normal detail.

7. Adjusting the parameterization: Once we have a new piercing point, we need to adjust the parameterization to reflect this. Essentially, the adjusted parameterization u should be such that the piercing point has the parameters $u(\mathbf{v}) =: u(\mathbf{q})$. When imposing such an isolated point constraint on the parameterization, there is no mathematical guarantee against flipping. Hence we draw a new piecewise linear curve through $u(\mathbf{q})$ in the parameter domain. This gives a new curve on the surface which passes through $\mathbf{q}$, see Figure 7. We then recompute the parameterization for each of the patches onto a triangle separately. We use a piecewise linear boundary condition with the half point at $\mathbf{q}$ on the common edge.

When all the new midpoints for the edges of a face of level j are computed, we can build the faces of level $j+1$. This is done by drawing three new curves inside the corresponding region of the original mesh, see Figure 8. Before that operation happens we need to ensure that a valid parameterization is available within the patch. The patch is parameterized onto a triangle with three piecewise linear boundary conditions each time putting the new points at the midpoint. Then the new points are connected in the parameter domain which allows us to draw new finer level curves on the original mesh. This produces a metamesh similar to [16], so that the new net of curves replicates the structure of the semi-regular hierarchy on the surface of the original. The construction of the semi-regular mesh can be done adaptively with the error driven procedure from MAPS [18]. An example of parameterization adjustment after two levels of adaptive subdivision is shown in the bottom middle of Figure 5. Note that as the regions for which we compute parameterizations become smaller, the starting guesses are better and the solver convergence becomes faster and faster.

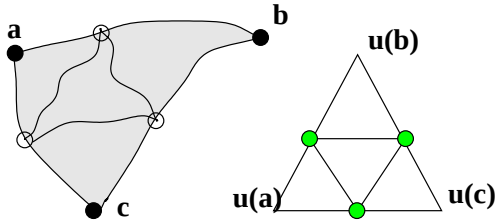


Figure 8: *Face split: Quadrisection in the parameter plane (left) leads to three new curves within the triangular patch (right).*

The aperture parameter κ of the piercing procedure provides control over how much of the original parameterization is preserved in the final mesh and consequently, how many non-normal details will appear. At $\kappa = 0$ we build a *non-normal* mesh entirely based on the original global parameterization. At $\kappa = 1$ we attempt to build a purely *normal* mesh independent of the parameterization. In our experience, the best results were achieved when the aperture was set low (0.2) at the coarsest levels, and then increased to 0.6 on finer levels. On the very fine levels of the hierarchy, where the geometry of the semi-regular meshes closely follows the original geometry, one can often simply use a naive piercing procedure without parameter adjustment.

One may wonder if the continuous readjustment of parameterizations is really necessary. We have tried the naive piercing procedure without parameterization from the base domain and found that it typically fails on all models. An example is Figure 9 which shows 4 levels of naive piercing for the torus starting from a 102 vertex base mesh. Clearly, there are several regions with flipped and self-intersecting triangles. The error is about 20 times larger than the true normal mesh.

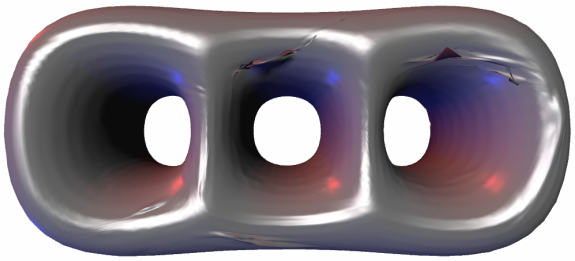


Figure 9: *Naive piercing procedure. Clearly, several regions have flipped triangles and are self-intersecting.*

Dataset	Size	Base	Normal mesh size	Not normal (%)	% L^2 error	Time (min)
Feline	49864	156	40346	729 (1.8%)	.015	4
Molecule	10028	37	9521	270 (2.8%)	.075	1.5
Rabbit	16760	33	8235	196 (2.4%)	.037	2
Torus3	5884	98	5294	421 (8.0%)	.03	3
Skull	20002	112	25376	817 (3.2%)	.02	2.5
Horse	48485	234	59319	644 (1.1%)	.004	6.8

Table 1: *Summary of normal meshing results for different models. The normal mesh is computed adaptively and contains roughly the same number of triangles as the original mesh. The relative L^2 errors are computed with the I.E.I.-CNR Metro tool. The times are reported on a 700MHz Pentium III machine.*

4 Results

We have implemented the algorithms described in the preceding section, and performed a series of experiments in which normal meshes for various models were built. The summary of the results is given in Table 1. As we can see from the table, the normal semi-regular meshes have very high accuracy and hardly any non normal details.

One interesting feature of our normal meshing procedure is the following: while the structure of patches comes from performing simplification there are far fewer restrictions on how coarse the base mesh can be. Note for example that the skull in Figure 1 was meshed with the tetrahedron as base mesh. This is largely due to the robust mesh parameterization techniques used in our approach.

Figure 10 shows normal meshes for rabbit, torus, feline, and skull, as well as close-up of feline (bottom left) normal mesh. Note how smooth the meshes are across global edges and global vertices. This smoothness mostly comes from the normality, not the parameterization. It is thus an intrinsic quantity.

One of the most interesting observations coming from this work is that locally the normal meshes do not differ much from the non-normal ones, while offering huge benefits in terms of efficiency of representation. For example, Table 2 shows how the "aperture parameter" κ that governs the construction of normal meshes affects the number of detail coefficients with non-trivial tangential components for the model of the three hole torus (these numbers are typical for other models as well). In particular, we see that already a very modest acceptance strategy ($\kappa = 0.2$) gets rid of more than 90% of the tangential components in the remeshed model, and the more aggressive strategies offer even more benefits without affecting the error of the representation.

5 Summary and Conclusion

In this paper we introduce the notion of *normal meshes*. Normal meshes are multiresolution meshes in which vertices can be found in the normal direction, starting from some coarse level. Hence only one scalar per vertex needs to be stored. We presented a robust

κ	normal	error (10^{-4})
0	0%	1.02
0.2	91.9%	1.05
0.4	92.4%	1.04
best	98.3%	1.02

Table 2: *The relation between the acceptance strategy during the piercing procedure and the percentage of perfectly normal details in the hierarchy. The original model has 5884 vertices, all the normal meshes have 26002 vertices (4 levels uniformly), and the base mesh contained 98 vertices. The best strategy in the last line used $\kappa = 0.2$ on the first three levels and afterward always accepted the piercing candidates.*

algorithm for computing normal semi-regular meshes of any input mesh and showed that it produces very smooth triangulations on a variety of input models.

It is clear that normal meshes have numerous applications. We briefly discuss a few.

Compression Usually a wavelet transform of a standard mesh has three components which need to be quantized and encoded. Information theory tells us that the more non uniform the distribution of the coefficients the lower the first order entropy. Having 2/3 of the coefficients exactly zero will further reduce the bit budget. From an implementation viewpoint, we can almost directly hook the normal mesh coefficients up to the best known scalar wavelet image compression code.

Filtering It has been shown that operations such as smoothing, enhancement, and denoising can be computed through a suitable scaling of wavelet coefficients [7]. In a normal mesh any such algorithm will require only 1/3 as many computations. Also large scaling coefficients in a standard mesh will introduce large tangential components leading to flipped triangles. In a normal mesh this is much less likely to happen.

Texturing Normal semi-regular meshes are very smooth inside patches, across global edges, and around global vertices even when the base domain is exceedingly coarse, cf. the skull model. The implied parameterizations are highly suitable for all types of mapping applications.

Rendering Normal maps are a very powerful tool for decoration and enhancement of otherwise smooth geometry. In particular in the context of bandwidth bottlenecks it is attractive to be able to download a normal map into hardware and only send smooth coefficient updates for the underlying geometry. The normal mesh transform effectively solves the associated inverse problem: construct a normal map for a given geometry.

The concept of normal meshes opens up many new areas of research.

- Our algorithm uses interpolating subdivision to find the base point. Building normal meshes with respect to approximating subdivision is not straightforward.
- The theoretical underpinnings of normal meshes need to be studied. Do continuous variable normal descriptions of surfaces exist? What about stability? What about connections with curvature normal flow which acts to reduce normal information?
- We only addressed semi-regular normal meshes here, while the definition allows for the more flexible setting of progressive irregular mesh hierarchies.
- Purely scalar compression schemes for geometry need to be compared with existing coders.
- Generalize normal meshes to higher dimensions. It should be possible to represent a M dimensional manifold in N dimensions with $N - M$ variables as opposed to the usual N.
- The current implementation only works for surfaces without boundaries and does not deal with feature curves. We will address these issues in our future research.

Acknowledgments This work was supported in part by NSF (ACI-9624957, ACI-9721349, DMS-9874082, DMS 9872890), Alias|Wavefront, a Packard Fellowship, and a Caltech Summer Undergraduate Research Fellowship (SURF). Special thanks to Nathan Litke for his subdivision library, to Andrei Khodakovsky, Mathieu Desbrun, Adi Levin, Arthur Olson, and Zoë Wood for helpful discussions, Chris Johnson for the use of the SGI-Utah Visual Supercomputing Center resources, and to Cici Koenig for production help. Datasets are courtesy Cyberware, Headus, The Scripps Research Institute, and University of Washington.

References

[1] CERTAIN, A., POPOVIC, J., DEROSE, T., DUCHAMP, T., SALESIN, D., AND STUETZLE, W. Interactive Multiresolution Surface Viewing. *Proceedings of SIGGRAPH 96* (1996), 91–98.

[2] COHEN, J., OLANO, M., AND MANOCHA, D. Appearance-Preserving Simplification. *Proceedings of SIGGRAPH 98* (1998), 115–122.

[3] COOK, R. L. Shade trees. *Computer Graphics (Proceedings of SIGGRAPH 84) 18*, 3 (1984), 223–231.

[4] DAUBECHIES, I., GUSKOV, I., AND SWELDENS, W. Regularity of Irregular Subdivision. *Constr. Approx. 15* (1999), 381–426.

[5] DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow. *Proceedings of SIGGRAPH 99* (1999), 317–324.

[6] DONOHO, D. L. Interpolating wavelet transforms. Preprint, Department of Statistics, Stanford University, 1992.

[7] DONOHO, D. L. Unconditional Bases are Optimal Bases for Data Compression and for Statistical Estimation. *Appl. Comput. Harmon. Anal. 1* (1993), 100–115.

[8] DYN, N., LEVIN, D., AND GREGORY, J. A. A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control. *ACM Transactions on Graphics 9*, 2 (1990), 160–169.

[9] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. *Proceedings of SIGGRAPH 95* (1995), 173–182.

[10] FLOATER, M. S. Parameterization and Smooth Approximation of Surface Triangulations. *Computer Aided Geometric Design 14* (1997), 231–250.

[11] GARLAND, M., AND HECKBERT, P. S. Surface Simplification Using Quadric Error Metrics. In *Proceedings of SIGGRAPH 96*, 209–216, 1996.

[12] GOLUB, G. H., AND LOAN, C. F. V. *Matrix Computations*, 2nd ed. The John Hopkins University Press, Baltimore, 1983.

[13] GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. Multiresolution Signal Processing for Meshes. *Proceedings of SIGGRAPH 99* (1999), 325–334.

[14] KHODAKOVSKY, A., SCHRÖDER, P., SWELDENS, W. Progressive Geometry Compression. *Proceedings of SIGGRAPH 2000* (2000).

[15] KRISHNAMURTHY, V., AND LEVOY, M. Fitting Smooth Surfaces to Dense Polygon Meshes. *Proceedings of SIGGRAPH 96* (1996), 313–324.

[16] LEE, A. W. F., DOBKIN, D., SWELDENS, W., AND SCHRÖDER, P. Multiresolution Mesh Morphing. *Proceedings of SIGGRAPH 99* (1999), 343–350.

[17] LEE, A. W. F., MORETON, H., HOPPE, H. Displaced Subdivision Surfaces. *Proceedings of SIGGRAPH 00* (2000).

[18] LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. MAPS: Multiresolution Adaptive Parameterization of Surfaces. *Proceedings of SIGGRAPH 98* (1998), 95–104.

[19] LEVOY, M. The Digital Michelangelo Project. In *Proceedings of the 2nd International Conference on 3D Digital Imaging and Modeling*, October 1999.

[20] LÉVY, B., AND MALLET, J. Non-Distorted Texture Mapping for Sheared Triangulated Meshes. *Proceedings of SIGGRAPH 98* (1998), 343–352.

[21] LOUNSBERY, M., DEROSE, T. D., AND WARREN, J. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *ACM Transactions on Graphics 16*, 1 (1997), 34–73. Originally available as TR-93-10-05, October, 1993, Department of Computer Science and Engineering, University of Washington.

[22] SCHRÖDER, P., AND SWELDENS, W. Spherical Wavelets: Efficiently Representing Functions on the Sphere. *Proceedings of SIGGRAPH 95* (1995), 161–172.

[23] ZORIN, D., AND SCHRÖDER, P., Eds. *Subdivision for Modeling and Animation*. Course Notes. ACM SIGGRAPH, 1999.

[24] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interpolating Subdivision for Meshes with Arbitrary Topology. *Proceedings of SIGGRAPH 96* (1996), 189–192.

[25] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interactive Multiresolution Mesh Editing. *Proceedings of SIGGRAPH 97* (1997), 259–268.

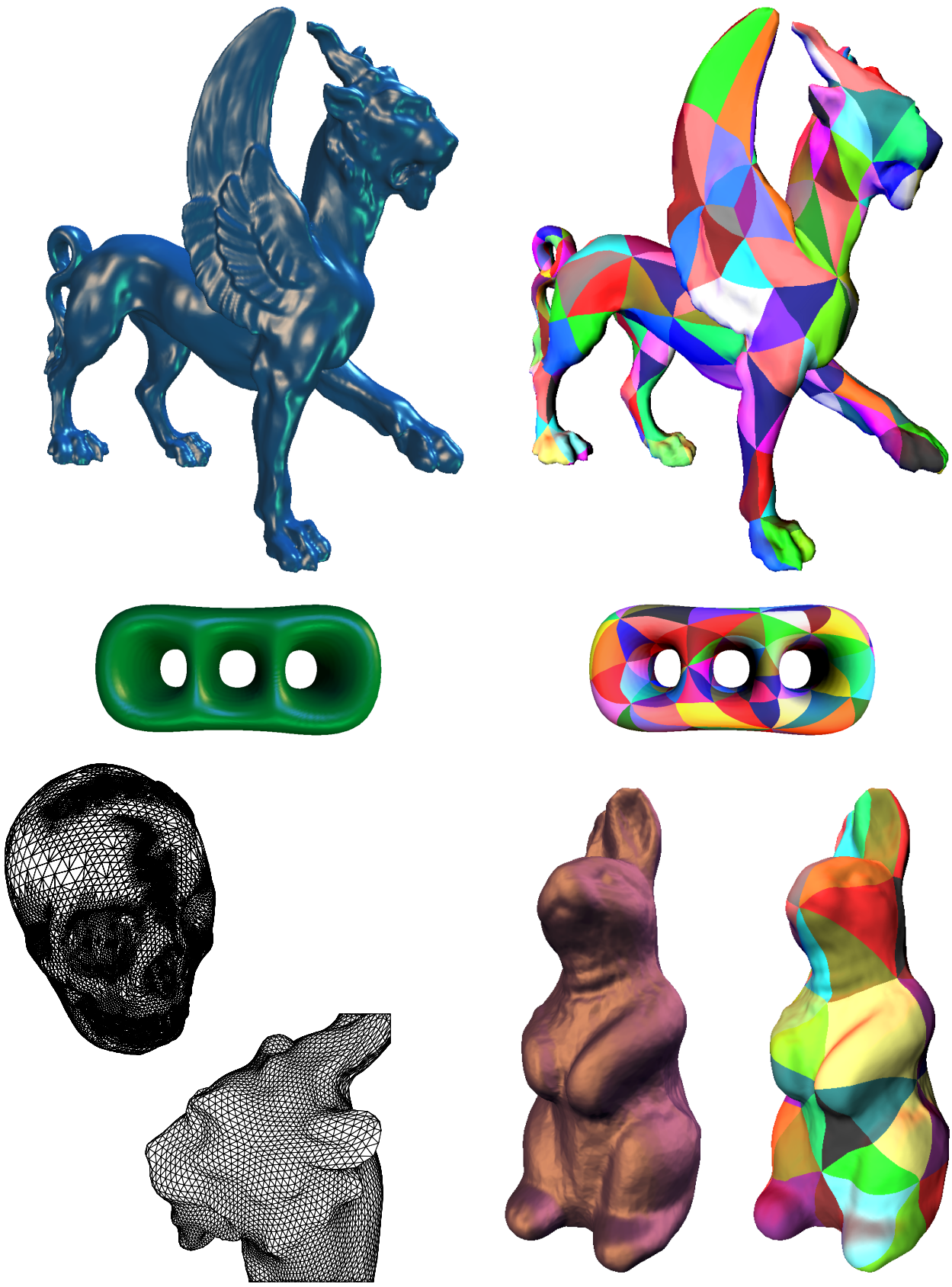


Figure 10: *Colorplate.*

Topological Noise Removal

Igor Guskov Zoë J. Wood

Caltech

Abstract

Meshes obtained from laser scanner data often contain topological noise due to inaccuracies in the scanning and merging process. This topological noise complicates subsequent operations such as remeshing, parameterization and smoothing. We introduce an approach that removes unnecessary nontrivial topology from meshes. Using a local wave front traversal, we discover the local topologies of the mesh and identify features such as small tunnels. We then identify non-separating cuts along which we cut and seal the mesh, reducing the genus and thus the topological complexity of the mesh.

Key words: Meshes, irregular connectivity, topology.

1 Introduction

Acquisition of computer models with highly detailed geometry is currently practical due to developments in laser range finder technology. Huge volumes of geometric data are routinely acquired and used in design, manufacturing, and entertainment. Raw irregular meshes coming from model acquisition contain millions of triangles, and require efficient processing tools. Such data is typically converted into a more efficient and "regular" representation such as NURBS or other spline/subdivision-based multiresolution surface representations. This process is called *remeshing* in the graphics literature [22][18] [21][25][24]. Several remeshing methods use simplification hierarchies of the initial irregular mesh in order to build efficient computational procedures. However, raw irregular meshes extracted from noisy volumetric data often have small tunnels and handles: artifacts of the acquisition process. We present an algorithm to eliminate such topological "noise", greatly improving the construction of the simplification hierarchy and thus in turn, improving the final remeshed model.

Scanned geometry can easily contain millions of data points, therefore the manual removal of artifacts is a tedious and time-consuming task; we would like to automate this process as much as possible. However, scanned models, such as mechanical parts, potentially have important non-trivial topology (holes, handles, tunnels, etc.). One therefore needs a clear criteria to discern which tunnels can be safely removed algorithmically. The contribution of this paper is to introduce a simple criteria for identifying topological noise, and a fast algorithm that finds small tunnels in the data, and removes them one by one. The user can control criteria to help determine which tunnels are noise and which are inherent to the model. In addition, we show that the performance of the naive implementation of our topology filtering algorithm can be significantly improved by a preprocessing step.

Figure 1: *Scanned meshes from Stanford 3D model repository [26]. All three meshes are valid 2-manifolds: the Buddha has genus 104, the dragon has genus 46, and David's head has genus 340. Most of these tunnels/handles are noise and can be safely removed.*

To demonstrate our approach we apply our technique to a variety of the Stanford laser range finder datasets. For example, we consider the dataset of the David's head from Stanford's Michelangelo project [26]. The original irregular mesh has genus 340. Obviously, none of these 340 tiny tunnels are actually present in the original

sculpture, therefore all these tunnels (or handles) can be removed to facilitate further processing tasks. An irregular mesh of David's head containing more than a million triangles is processed by our algorithm in one hour, removing 313 (92%) of the tunnels automatically. The algorithm can also be run in an interactive mode leaving the decision to remove bigger handles to a user. Complete filtering can also be made efficient using a combined filtering and simplification approach (see section 3 for more details).

1.1 Setting

The main application of our method is the processing of meshes coming from 3D model acquisition such as laser scanning. During acquisition, a complex model is often built from several scans. Each scanning pass produces a grid of points in space possibly with holes. A number of popular mesh reconstruction methods [9][31] [19] combine several range maps using an auxiliary volumetric representation: a signed distance volume constructed from a collection of scans. An isosurface is then extracted using the Marching Cubes algorithm [27]. The result is an irregular mesh that is a proper manifold with boundary. The data coming from the scanner can be noisy and incomplete, hence the noise in the signed distance volume.

While the manifold property of the extracted surface can be guaranteed with small modifications to the original Marching Cubes algorithm[23], we observe that for noisy data it still produces topological artifacts such as tiny handles. It is often important to remove these handles so that they do not encumber later processing, such as simplification [20], smoothing, denoising [10], and remeshing. Figure 8 shows the result of applying a smoothing procedure to a mesh with handles. While most of the surface gets smoother, the areas containing handles have visible artifacts.

The effect of small handles on simplification algorithms requires a more careful explanation. One can distinguish two classes of simplification procedures: the algorithms of the first class assume that the original mesh is a proper manifold with a boundary and preserve the genus of the surface for each simplification step. Such simplification is often used as an integral part of some larger multiresolution processing procedure that may rely on the topological equivalence of meshes on different levels of the hierarchy [25][8][17] [35][18]. This kind of simplification algorithm will clearly benefit from topological noise removal – we show several examples of such improvements in Section 3. The simplification algorithms of the second class (e.g. [15]) do not assume the manifold property and will therefore simplify the given mesh (or polygon/edge soup, or simplicial complex[30]) without noticing small tunnels and handles. However, these algorithms are useful only for simplification per se, and despite starting with a proper manifold they cannot guarantee that the manifold property will be maintained for coarser levels of the hierarchy.

1.2 Related work

A variety of researchers have relied on coding or matching the topology of a given mesh to a new configuration [29][3][4]. Most recently a lot of attention has been directed towards general simplicial complexes. Specifically, the problem of preserving the topology of simplicial complexes while applying edge contractions was considered by Dey et al [34]. The recent paper by Edelsbrunner et al.[11] considers topological simplification in the context of alpha complexes. It is worth noting that topological simplification of simplicial complexes in $\mathbf{R}^3$ is a much harder and less intuitive problem than the one we consider.

El-Sana and Varshney [12][13] address a similar problem of controlled topology simplification for polygonal models. Their approach identifies removable tunnels by rolling a sphere of small radius over the object and filling the tunnels that are not accessible. The method performs well for mechanical CAD models. The interaction between mesh and topology simplification is also considered. Our approach is different in that it identifies tunnels by working within the surface, and thus can be applied to self-intersecting meshes as long as they are topologically 2-manifolds. Also, the focus of our work is to identify very small tunnels in noisy meshes.

Work has been done by Stander et al [33] on using critical points from Morse theory to guarantee the topology of the polygonization of an implicit surface. It is difficult to generalize this work to the irregular mesh setting without becoming computational intensive (see [6] for a potential solution). We focus on discrete methods that can rapidly discover the topology.

Recent work by Wood et al [37] presented an algorithm to quickly identify and reconstruct the topology of a surface implicitly represented in a volume. This work uses a wave front traversal in order to identify the global topology of the surface. The algorithm presented here has similarities but is generalized to the mesh setting with optimizations to discover small local topology and with optimizations to identify topological events. This work is closely related to work done by Axen [7] which relates a discrete wavefront traversal and critical points from Morse theory.

It is worth noting that there is another way to potentially "filter" or smooth the noisy topology of scanned data by smoothing/down-sampling the initial volume data. Although this approach may remove many of the

small tunnels present in the data, it will do so in an uncontrolled manner and will potentially wipe out other features of the model (thin tubes and connected components could be broken apart and the finer detailed geometry will disappear). Recent work by Gerstner and Pajarola [16] on topology preserving volume simplification is one potential solution to try to control the effect of the downsampling, however, presently this work offers no method to distinguish important topology inherent to the model (such as a large handle) and small tunnels.

Finally, a great deal of work has been focused on simplifying meshes in general. Work by both Popovic and Hoppe [30] and Garland and Heckbert [15] could be applied to simplify "away" the small noisy tunnels present in the scanned meshes. However, in our work we seek more explicit topology changes that can be potentially adapted in a multiresolution processing algorithms such as MAPS [25].

1.3 Overview of the algorithm

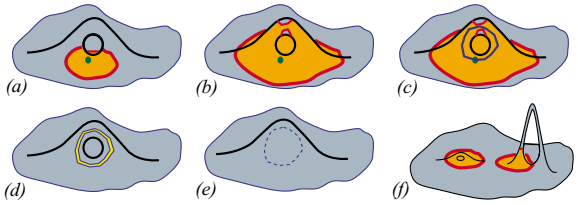


Figure 2: *(a-e) Overview of the algorithm: (a) a small region is grown around a seed face; (b) the genus of the grown region becomes non-zero; (c) a non-separating cut is found; (d) the mesh is cut; (e) both new holes are sealed. (f) The left handle is "fully inside" a ball of a small radius; the right handle is not. Note that both handles could be eliminated by short cuts. Our algorithm will only remove the left handle. (Formally, the left highlighted region is of genus one, while the right highlighted region is of genus zero.)*

We follow an approach similar to the ones presented in Wood et al. [37] and Axen et al.[7]. We grow an open region by adding faces one by one, while explicitly maintaining the active front edges. Every time a boundary component of the growing region touches itself along an edge, we split this boundary into two smaller boundary fronts and continue propagation. This results in a tree of active front components. Whenever boundaries of two different components touch along an edge, we claim to have found a handle. There are two stopping criteria for our region growing procedure – we either exhaust all the faces that are closer than some given radius from the seed face, or we actually find a handle in which case the growing stops and the mesh is cut along a non-separating curve. This operation does not change the connectedness of the surface but does reduce its genus introducing two new holes (boundaries), which are later triangulated using methods described in [32][25], or commercial packages [1][2]. (Figure 2 illustrates this process.) In this way, we remove the small handles one by one, filtering the local topology of the mesh.

2 Algorithm

We consider a triangular mesh $\mathcal{M} = (\mathcal{K}, \mathbf{x})$ where $\mathcal{K} = \mathcal{V} \cup \mathcal{E} \cup \mathcal{F}$ is an abstract simplicial complex representing the connectivity of the mesh ($\mathcal{V}$, $\mathcal{E}$, and $\mathcal{F}$ are sets of vertices, edges, and faces, correspondingly), and $\mathbf{x} : \mathcal{V} \to \mathbf{R}^3$ is the coordinate function that gives the coordinates of every vertex of $\mathcal{V}$. $\mathbf{x}$ can be extended to the polytope $|\mathcal{K}|$ of $\mathcal{K}$ using barycentric coordinates [28]. In this paper the focus is on meshes extracted as isosurfaces of certain volumetric functions, and therefore, such meshes are guaranteed to be oriented manifolds. Thus, all meshes considered in this paper are presumed to be oriented manifolds with boundary. Topology of such surfaces is easily characterized by their genus.

2.1 m-Closures

Our interest lies in finding "small" tunnels in the mesh, where the "smallness" will be defined later. Thus, we need to characterize topological properties of local regions of the mesh. For example, given a collection of faces $T = \{t_1, t_2, \ldots, t_k\}$ we would like to explore topological properties of the surface region defined by this set of faces. One way to approach this characterization would be to find the closure $\bar{T}$ in $\mathcal{K}$, and look at its properties. Note that the closure $\bar{T}$ for arbitrary T may not have the manifold property anymore, see Figure 3 for example. It is in fact a subcomplex of $\mathcal{K}$ and can be characterized as a general 2-complex, see [36]. However, that characterization is far too general for our purposes here. We therefore introduce a different "closure" operation that for a mesh region builds a corresponding mesh that is a manifold with boundary, and as such can be easily described by its genus. We call this operation *manifold closure* or simply *m-closure*, defined as follows.

First, note that Figure 3 represents the only way that $\bar{T}$ can be non-manifold. Moreover, it can be fixed with the following procedure (see Figure 3 for an illustration): for every non-manifold vertex $v \in \bar{T}$ its star neighborhood in $\bar{T}$ can be written as union of a number of semi-stars $H_v^{(i)}$: $St_{\bar{T}} v = \bigcup_{i=1}^{N_v} H_v^{(i)}$, where $\bigcap_{i=1}^{N_v} H_v^{(i)} = \{v\}$, and each semi-star is of the form: $H_v^{(i)} = \{\{v\}, \{v, u_0\}, \{v, u_0, u_1\}, \ldots, \{v, u_{k-1}, u_k\}, \{v, u_k\}\}$.

We define the *m-closure* of T in $\mathcal{K}$ as the mesh obtained from $\bar{T}$ by splitting every non-manifold vertex v with N_v adjacent semi-stars into N_v vertices $v^{(1)}, \ldots, v^{(N_v)}$, and replacing each occurrence of v in the simplices of $St_{\bar{T}} v$

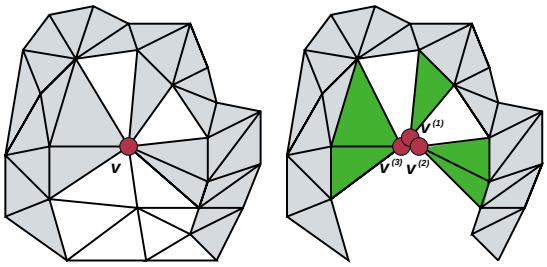


Figure 3: *Non-manifold closure ($N_v = 3$) is fixed by making three copies of the vertex v.*

by the appropriate new vertex depending on which semi-star they belong to. We denote the resulting mesh as $\bar{m}T$. Note that the interiors of m-closure and usual closure coincide: $int\,\bar{T} = int\,\bar{m}T$.

2.2 Small tunnels

It is now necessary to define which tunnels need to be removed. For that purpose, we consider the dual graph $(\mathcal{F}, \mathcal{E}')$ of the mesh $\mathcal{M}$ where a dual edge (t_1, t_2) between two faces of the mesh is in $\mathcal{E}'$ if t_1 and t_2 share a (primal) edge in the triangulation. If some non-negative weight function w is defined on $\mathcal{E}'$, we can now define the distance $d(s,t)$ between any two faces s and t as the minimal sum of weights over all the paths in the dual graph. One easy example is given by setting $w(e') = 1$ for every $e' \in \mathcal{E}'$. It is also possible to make weights that would approximate geodesic distances on a manifold. In this paper we use $w \equiv 1$.

Now we can give the general principle that we use to remove small tunnels:

ε-simple meshes

Mesh $\mathcal{M}$ is ε-simple if for every face $t \in \mathcal{F}$ the m-closure of dual ε-ball $\bar{m}\{s : d(s,t) < \varepsilon\}$ is of genus zero.

Our goal therefore becomes to convert a given mesh into an ε-simple mesh. This can be done by finding closed cuts that leave the mesh connected. Each such cut will reduce the genus of the surface by one. In the following sections, we introduce an algorithm to find such non-separating closed cuts (a cut is non-separating if it leaves the surface connected [5].) These cuts will be found inside the corresponding ε-balls; note however that such short non-separating cuts can exist in meshes that are ε-simple for small ε, such as the ones containing long narrow handles, see Figure 2(f). However, it is not clear that such long handles should be automatically removed. In our approach, we will only find cuts corresponding to handles that are completely contained in small regions of the mesh.

2.3 Region growing

In this section we describe an algorithm that looks for tunnels in the neighborhood of a seed face. Later, in Section 2.6 we explain a global search for tunnels that will use this local procedure as an elementary operation.

The local procedure starts with a seed face $t_{seed} \in \mathcal{F}$. The faces from the ε-ball around t_{seed} are considered one by one in the order produced by using Dijkstra's algorithm on the dual graph. Thus, a sequence $t_1, t_2, \ldots, t_k$ is constructed. We define the i-th active region as $A_i(t_{seed}) := \bar{m}\{t_{seed}, t_1, \ldots, t_i\}$ for $i = 1, \ldots, k$. Algorithmically, the active region is grown one face at a time, while the explicit representation of active boundaries is maintained. Every time a new face is added, we check the genus of the resulting active region. The process starts with one triangle which is obviously of genus zero. We then proceed either until all the faces of the ε-ball are exhausted, or until we find that after the current triangle is added, the genus of the active region has grown. If the latter happens, the region growing stops and a non-separating closed cut is found inside the active region. We then cut the mesh (possibly locally subdividing it), seal the two resulting holes, and start with the current seed face again. Thus, the small tunnels in the mesh are extinguished one by one.

We now describe the particulars of maintaining the active region and tracking its genus.

2.4 Evolution of the active region

Suppose the active region A is given and another face t needs to be added to it. By construction, $A \cap t$ contains an edge. The change of active region is performed using the following three operations: *add-triangle*, *close-crack*, and *merge-edge*[1] . We describe these operations below in more detail.

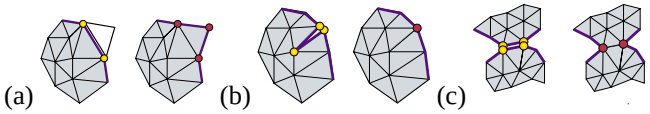


Figure 4: *(a)* add-triangle *operation; (b)* close-crack *operation; (c)* merge-edge *operation. Current mesh region shown in gray, with its boundary in blue.*

Add-triangle

We assume that the active region and the new incoming triangle share at least one common edge. Then the *add-triangle* operation adds the triangle to the active region by merging across a common edge. The resulting mesh has one more face, two more edges, and one more vertex than

[1]Note that there is no need for a *merge-vertex* operation (when a single vertex is adjacent to more than two boundary edges) due to *m-closure*.

the original one (see Figure 4(a)). The number of boundary components does not change. Thus, the genus of the corresponding mesh region does not change. Indeed,

$$\begin{aligned}\chi_{new} &= V_{new} - E_{new} + F_{new} + H_{new} \\ &= (V_{old}+1) - (E_{old}+2) + (F_{old}+1) + H_{old} \\ &= \chi_{old}.\end{aligned}$$

Since the genus of the region is $g = 1 - \chi/2$, and χ is unchanged, the genus of the current mesh region is preserved during the *add-triangle* operation.

In order to find the non-intersecting cut later, each face stores a pointer to the face to which it was added. To set up the notation, let t be the new face and $t' \in A$ be a face from the active region that shared a common edge with t. We call t' the *parent of* t, or $t' = parent(t)$.

Close-crack

Once the new triangle is added to the mesh we need to resolve possible self-adjacencies along the boundary. One local inconsistency is depicted in Figure 4(b). We fix the boundary locally by eliminating two boundary edges. The resulting mesh has one less edge, and one less vertex than the original one. The number of faces and boundary components does not change. Thus, the genus of the corresponding mesh does not change. Again,

$$\begin{aligned}\chi_{new} &= (V_{old}-1) - (E_{old}-1) + F_{old} + H_{old} \\ &= \chi_{old}.\end{aligned}$$

Merge-edge

The last operation required to maintain a consistent active region is not local, in that it requires adjacency tests between different parts of the boundaries, or even between different boundary components. Indeed, the *close-crack* operation cannot resolve situations such as the one shown in Figure 4(c). Here two edges lying on two separate pieces of the boundary of the current region correspond to the same edge of the original mesh. We fix this inconsistency by merging the current region(s) across this edge. As a result the number of boundary components will either increase by one (when the merged edges belong to the same boundary component), or decrease by one (when two different boundary components become one). Note that these two cases closely correspond to the topological events described in [37], when the active edge front either splits into two when a handle in the surface is encountered, or when it merges back into a single front at the other side of the handle. The *merge-edge* operation results in one less edge and two less vertices for the active region, and the number of faces does not change. Depending on the value of the change in the number of boundary components we will encounter two cases:

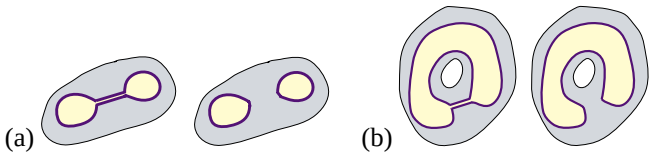


Figure 5: *Before and after* merge-edge *operation: (a) A boundary component splits; (b) two boundary components merge. Current mesh region shown in gray, its boundary in blue, the unexplored region of the mesh is in yellow.*

A boundary splits. Figure 5(a).

$$\begin{aligned}\chi_{new} &= (V_{old}-2) - (E_{old}-1) + F_{old} + (H_{old}+1) \\ &= \chi_{old}.\end{aligned}$$

Boundaries merge. Figure 5(b).

$$\begin{aligned}\chi_{new} &= (V_{old}-2) - (E_{old}-1) + F_{old} + (H_{old}-1) \\ &= \chi_{old} - 2.\end{aligned}$$

In this last case the genus of the active region increases by one. When this final case is detected, we proceed by performing a non-separating cut, thus reducing the genus by one.

2.5 Cutting the mesh

In this section, we describe how a non-separating cut is found inside the active region after a merge-edge operation has merged two boundary components. Suppose that the two boundaries merged along the edge $e_M = \{v^{(1)}, v^{(2)}\} = t^{(1)} \cap t^{(2)}$. We build two sequences of faces, $p^{(1)}$ and $p^{(2)}$, defined as $p^{(j)} = (t_1^{(j)}, \ldots, t_{K^j}^{(j)})$, where $t_{k+1}^{(j)} = parent(t_k^{(j)}), j = 1, 2$. Note that both of these face paths end at the original seed face which has no parent. After excluding a common tail of these two paths we have a closed path in the dual graph of the active region. It is then possible to subdivide the faces on this closed path so that there is a closed cut along the edges of this locally subdivided mesh which does not intersect itself, see Figure 6. Note that this path is *completely inside the interior of the current active mesh region*.

We can also prove that this cut is non-separating, that is, it leaves the active mesh region (and hence the mesh itself) connected. In order to prove that we simply notice that the two vertices $v^{(1)}$ and $v^{(2)}$ lie on the different sides of the cut locally but we can reach $v^{(2)}$ from $v^{(1)}$ by following the boundary of the current active region (we can do that because the cut is fully inside the active region and thus does not touch the boundary). We also further reduce the length of the cut, by using reductions similar to the one shown in Figure 6. During these reductions we do not allow faces $t^{(1)}$ and $t^{(2)}$ to disappear, therefore the

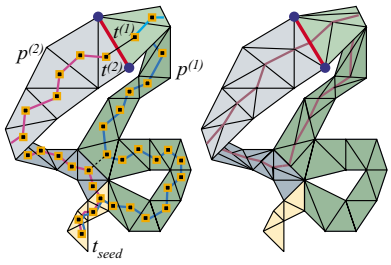


Figure 6: *Cutting the mesh. Left: two paths in the dual graph from the faces $t^{(1)}$ and $t^{(2)}$ to the seed face are found by following the parent links. Note that the closed face path in the dual graph can be reduced as shown by the black dashed line (two adjacent faces allow a shorter connection rather than taking the longer path through the first common face of the paths $p^{(1)}$ and $p^{(2)}$). Right: the non-separating cut with the corresponding local mesh refinement.*

argument above still holds. We then seal these two new gaps in the mesh, and thus remove the handle.

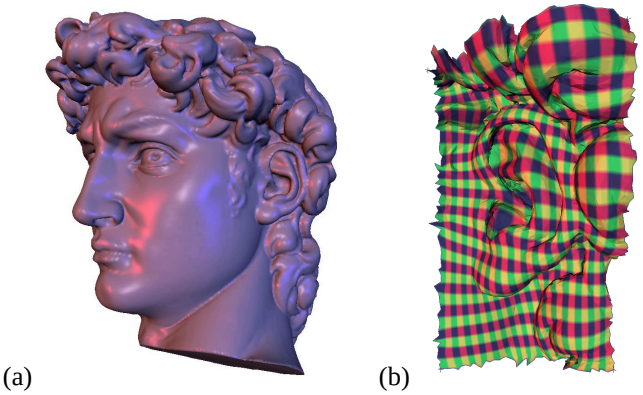
The subdivision performed during the cut computation changes distances in the dual graph. We fix this problem by assigning zero weights to the new edges introduced during subdivision (of course, the dual edges corresponding to the edges in the cut itself simply disappear from the dual graph of the modified mesh.)

2.6 Global procedure and preprocessing

In the previous section we described a procedure that grows a mesh region of some radius $\varepsilon > 0$ centered at a seed face and removes all the tunnels that are discovered inside this mesh region one by one. We can run this procedure starting from all the faces in the original mesh. This will produce a mesh that is ε-simple. However, as ε grows the running times of this naive algorithm become unacceptable. We propose a preprocessing step that excludes large portions of seed faces from the consideration. We rely on the following fact which is true in a metric space. Let $B_R(t_0)$ be the closed ball of radius R centered at t_0 (note that we measure the distances on the surface, so in our case, a ball is a surface region.) Then for any $t' \in B_{R-\varepsilon}(t_0)$ the ball centered at t' of radius ε is contained in $B_R(t_0)$, in fact, $B_\varepsilon(t') \subset B_R(t_0)$. Therefore, in the preprocessing step we will be growing balls until their genus changes, without any restriction on their radius. Suppose that we have grown a mesh region A that includes the ball $B_R(t_0)$ for some $R > \varepsilon$, and the genus of A is zero. Then we can be assured that any subset of A will also be of genus zero, and since the balls of radius ε centered inside the smaller region $B_{R-\varepsilon}(t_0)$ are subsets of A, we can exclude them from the potential seed set. These large regions are seeded in the preprocessing step at randomly chosen faces of the original mesh (in practice, taking one percent of the original number of faces produces good results). This procedure greatly reduces the potential seed set for a given ε. For example, without preprocessing, the algorithm takes 1147 seconds to perform filtering with radius 3 on the David's head model; while the improved procedure takes only 136 seconds. More performance numbers can be found in Table 1.

3 Results

We have implemented the algorithm described in the preceding sections and performed various experiments reducing the topological noise for a number of meshes from the Stanford Archive [26]. We have found that most of the models reconstructed using Curless and Levoy's VRIP method [9] have topological artifacts. We noticed that meshes that were more convoluted in shape typically have more tiny tunnels than the simple shaped models. In addition, the presence of topological noise is more frequent in the higher resolution models. We have run our algorithm on models of different resolution with different threshold radius settings and recorded the number of tunnels removed and the algorithm's running time. These results are illustrated in Table 1.



(a) (b)

Figure 7: *(a) The model of David's head is remeshed after topological noise has been removed. (b) The ear region can be easily parameterized onto a square after its topology has been filtered.*

We have applied various mesh processing techniques to meshes that have been topologically filtered using our algorithm with encouraging results. In particular, we were able to apply the multiresolution remesher of Guskov et al. [18] to the simplified genus zero mesh of David's head. The resulting remesh is shown in Figure 7. The base mesh for this remesh contains 262 triangles. It would be impossible to achieve such a small number of patches without first applying a topology filtering operation to the original data (remember that the original mesh had 340 tunnels).

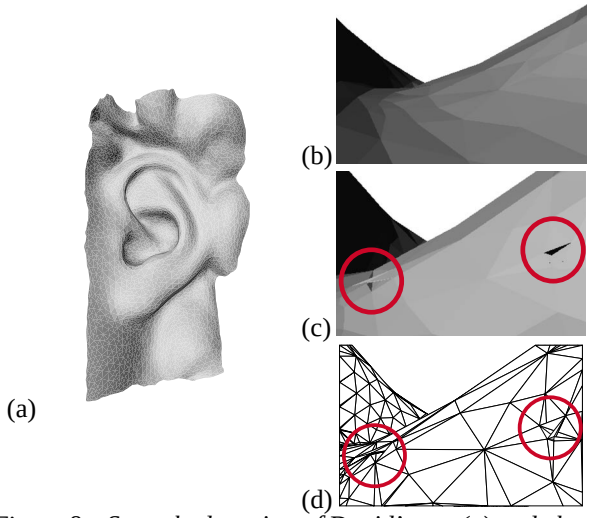


Figure 8: *Smoothed version of David's ear (a) and close up view of the smoothed ear after topology filtering (b) and a close up of the artifacts that occur without filtering (flipped triangles) (c) and a detailed view of the tunnels causing the artifacts (d).*

Similarly, parameterization of mesh regions is a fundamental part of many remeshing, texturing, and other mesh processing algorithms. The Figure 7 shows the parameterized mesh region of the David's ear. The texture coordinates are assigned with the (u, v)-coordinates computed with the shape-preserving parameterization of Floater [14]. The original unfiltered region of this mesh contained twelve tunnels and could not be properly parameterized onto the unit square. Our algorithm removes all of these tunnels in fifteen seconds, and produces a mesh that is homeomorphic to a square, allowing it to be parameterized.

Additionally, acquired meshes often contain geometric noise, and have to be filtered with various mesh smoothing/noise removal techniques. In particular, we used the method described in Desbrun et al. [10]. If the original mesh contains unnecessary non-trivial topological artifact, the smoothing procedure typically results in a mesh with artifacts that foil its appearance (such as flipped triangles), as shown in Figure 8. This is due to the fact that smoothing operators cannot modify the topology of the mesh, and the presence of these small handles impairs the smoothing process by limiting its effects. Attempts to smooth the region around small tunnels can potentially result in collapsing the tunnel, creating undesirable degeneracies. Thus first removing the topological noise greatly improves the performance of geometric noise removal procedures, as illustrated by Figure 8.

Finally, we have explored iterating between removing topological noise and applying topology preserving mesh

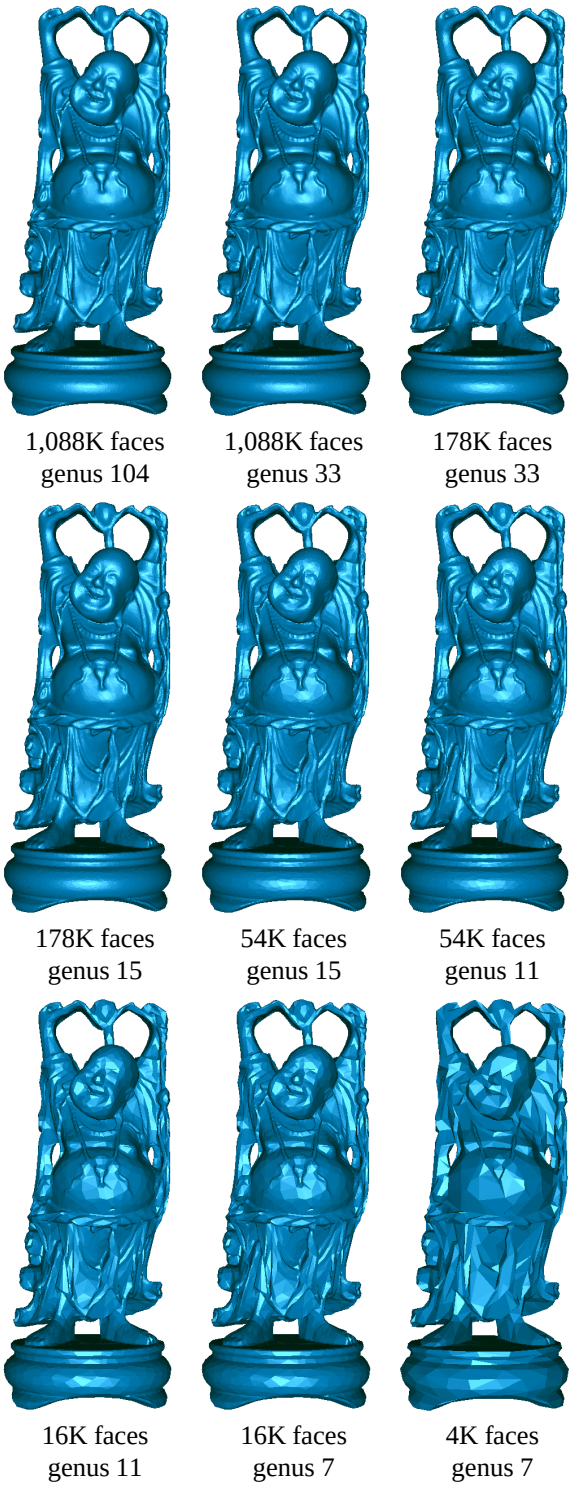


Figure 9: *Using both topology and geometry simplification on the Buddha mesh (see Table 9.) Note that all the meshes shown here are valid manifolds. The geometry simplification was performed with Raindrop Geomagic Studio [1]*

.

Dataset	Radius	Removed handles	Time
David's	4	55	10m 18s
head I	6	191	17m 22s
4000K faces	8	241	35m 34s
genus 340	10	264	1h 24m 43s
	12	283	3h 13m 30s
David's	4	193	3m 1s
head II	6	259	6m 1s
1173K faces	8	291	12m 53s
genus 340	10	301	27m 37s
	12	313	56m 52s
David's	4	286	1m 2s
head III	6	317	1m 58s
184K faces	8	323	4m 27s
genus 340	10	326	9m 36s
	12	330	19m 6s
David (complete	6	7	27m 3s
statue)	8	12	34m 4s
8254K faces	10	13	45m 11s
genus 20	12	14	57m 43s
Buddha	6	60	4m 16s
1087K faces	8	71	10m 23s
genus 104	10	82	34m 24s
	12	85	2h 43m 9s
Dragon	8	21	6m 4s
870K faces	10	32	16m 59s
genus 46	12	35	53m 3s
St.Matthew	6	3	21m 19s
3382K faces	12	4	29m 37s
genus 5			

Table 1: *Timings for topological noise removal are given for Pentium III Xeon 550 MHz.*

Size(faces)	Genus before	Genus after	Time
1087K	104	33	623s
178K	33	15	94s
54K	15	11	64s
8K	11	7	32s

Table 2: *Multiple resolutions processing of the Buddha mesh. Mesh simplification was used to reduce the face count of the models between the topology filtering steps (see Figure 9.) Threshold radius was set to 8 for all runs.*

simplification. Running these two processes alternatively decreases the amount of time to discover all the small tunnels on a given mesh. The results of such an iteration sequence are presented in Table 2 (the corresponding sequence of meshes is shown in Figure 9). It is clear that if the topology simplification is used as a part of a multiresolution technique such as remeshing, this gradual approach would be preferable for efficiency reasons. We leave the complete exploration of these ideas as future work.

4 Conclusions and future work

Topological noise is a serious problem for many scanned models. This noise results in visible artifacts when these meshes are smoothed, encumbers parameterization and hinders the performance of many multiresolution techniques. We have presented a simple criteria for identifying such topological noise and a computational procedure that removes these topological artifacts. The algorithm is very robust and is able to process extremely large meshes.

In this paper we have focused on removing the topological noise from the original resolution of the model and did not concern ourselves with larger scale genus changing operations. In fact most of the tunnels removed with our algorithm are in the very crooked parts of small regions of the mesh, and their removal does not affect the visual appearance of the model. It would be very interesting to explore genus changing operations in the multiresolution setting, perhaps directly within a mesh simplification or remeshing framework. Another exciting prospect for future work is the direct removal of topological noise from the original volume data.

5 Acknowledgments

This work was supported in part by DARPA and NSF through Caltech's OPAAL project (DMS-9875042), NSF (ACI-9721349, ACI-9982273, DMS-9872890, ASC-8920219), Packard Fellowship, Alias|Wavefront, and Intel. We would like to thank Mathieu Desbrun, Peter Schröder, Wim Sweldens, and Andrei Khodakovsky for many useful discussions about this work and we'd like to thank the Stanford Computer Graphics Group for sharing their wonderful models with us. Datasets of David's head, David statue, and St.Matthew are courtesy Digital Michelangelo Project, Stanford University.

6 References

[1] Geomagic studio 3.0, 2000. Raindrop Geomagic.

[2] Paraform 2.0, 2000. Paraform Inc.

[3] Ergun Akleman and Jianer Chen. Guaranteeing 2-manifold property for meshes. In *Proc. of Int. Conf. on Shape Modeling and Applications*, 1998.

[4] Ergun Akleman, Jianer Chen, and Vinod Srinivasan. A new paradigm for changing topology of 2-manifold polygonal meshes. In *Pacific Graphics'2000*, 2000.

[5] P. Aleksandrov. *Combinatorial Topology*, volume 1. Graylock Press, 1956.

[6] U. Axen. Computing morse functions on triangulated manifolds. In *Proceedings of the SIAM Symposium on Discrete Algorithms (SODA)*, 1999.

[7] U. Axen and H. Edelsbrunner. Auditory morse analysis of triangulated manifolds. In H.-C. Hege and K. Polthier, editors, *Mathematical Visualization*, pages 223–236. Springer-Verlag, Berlin, Germany, 1998.

[8] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-preserving simplification. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, pages 115–122, 1998.

[9] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. *Proceedings of SIGGRAPH 96*, pages 303–312, 1996.

[10] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. *Proceedings of SIGGRAPH 99*, pages 317–324, 1999.

[11] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. In *The 41st Ann. Symp. on Foundations of Comp. Science*, 2000.

[12] J. El-Sana and A. Varshney. Controlled simplification of genus for polygonal models. *Proceedings of the IEEE Visualization '97*, pages 403–412, 1997.

[13] J. El-Sana and A. Varshney. Topology simplification for polygonal virtual environments. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):133–144, June 1998.

[14] Michael S. Floater. Parameterization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14:231–250, 1997.

[15] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 96*, pages 209–216, 1996.

[16] Thomas Gerstner and Renato Pajarola. Topology preserving and controlled topology simplifying multiresolution isosurface extraction. *Visualization* '00 Proceedings , 2000.

[17] Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. *Proceedings of SIGGRAPH 99*, pages 325–334, 1999.

[18] Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schröder. Normal meshes. *Proceedings of SIGGRAPH 2000*, pages 95–102, 2000.

[19] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Computer Graphics (SIGGRAPH 1992 Proceedings)*, pages 71–78, 1992.

[20] Hugues Hoppe. Progressive meshes. *Proceedings of SIGGRAPH 96*, pages 99–108, 1996.

[21] L. Kobbelt, J. Vorsatz, U. Labsik, and H-P. Seidel. A shrink wrapping approach to remeshing polygonal surfaces. *Computer Graphics Forum*, 18:119 – 130, 1999.

[22] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. *Proceedings of SIGGRAPH 96*, pages 313–324, 1996.

[23] J.-O. Lachaud. Topologically Defined Iso-surfaces. Research Report 96-20, Laboratoire de l'Informatique du Parallélisme, ENS Lyon, France, 1996.

[24] A. Lee, H. Moreton, and H. Hoppe. Displaced subdivision surfaces. In *Computer Graphics (SIGGRAPH 2000 Proceedings)*, pages 85–94, 2000.

[25] Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. *Proceedings of SIGGRAPH 98*, pages 95–104, 1998.

[26] Marc Levoy. The digital michelangelo project. In *Proceedings of the 2nd International Conference on 3D Digital Imaging and Modeling*, Ottawa, October 1999.

[27] W. Lorensen and H. Cline. Marching cubes: A high resolution 3d surface reconstruction algorithm. In *Computer Graphics (SIGGRAPH 1987 Proceedings)*, pages 163–169, 1987.

[28] J. R. Munkres. *Elements of Algebraic Topology*. Addison-Wesley, Redwood City, 1984.

[29] Tamal Dey Nina Amenta, Sunghee Choi and Naveen Leekha. A simple algorithm for homeomorphic surface reconstruction. *ACM Symposium on Computational Geometry*, pages 213–222, 2000.

[30] Jovan Popovic and Hugues Hoppe. Progressive simplicial complexes. *Proceedings of SIGGRAPH 97*, pages 217–224, 1997.

[31] K. Pulli, T. Duchamp, H. Hoppe, J. McDonald, L. Shapiro, and W. Stuetzle. Robust meshes from multiple range maps. In *Proceedings of International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, pages 205–211, May 1997.

[32] J. R. Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Workshop on Applied Computational Geometry (Philadelphia, Pennsylvania)*, pages 124–133. Association for Computing Machinery, May 1996.

[33] Barton T. Stander and John C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 279–286, August 1997.

[34] S. Guha T. K. Dey, H. Edelsbrunner and D. V. Nekhayev. Topology preserving edge contraction. *Publ. Inst. Math. (Beograd) (N.S.)*, 66:23–45, 1999.

[35] Gabriel Taubin, André Gueziec, William Horn, and Francis Lazarus. Progressive forest split compression. *Proceedings of SIGGRAPH 98*, pages 123–132, 1998.

[36] E. F. Whittlesey. Finite surfaces: a study of finite 2-complexes. *Math. Mag.*, pages 11–22 and 67–80, 1960.

[37] Zoë Wood, Mathieu Desbrun, Peter Schröder, and David Breen. Semi-regular mesh extraction from volumes. In *Proceedings of Visualization 2000*, 2000.

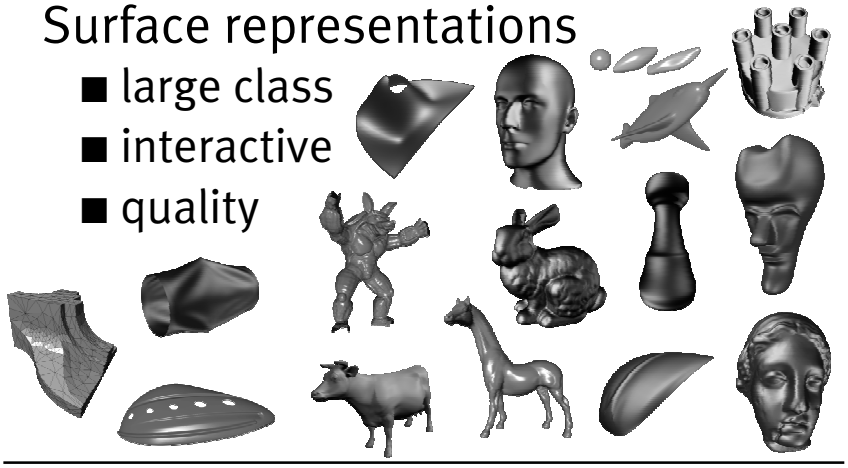
Multiresolution Subdivision Surfaces

Peter Schröder
Caltech

Modeling Geometry

Surface representations

- large class
- interactive
- quality



Geometric Modeling

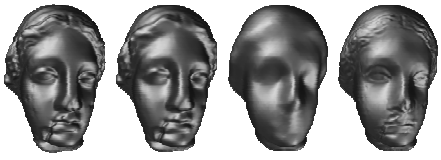
Common approaches

- polyhedral meshes
- spline (NURBS) patches
- solid modeling
- implicit surfaces
- generative (functional composition)

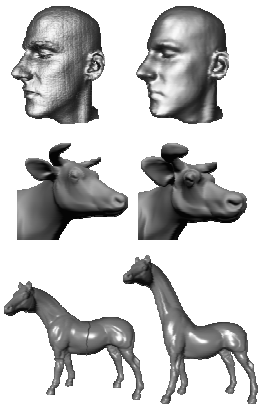
Geometry Processing

Signal processing tool box

- denoising,…
- enhancement,…
- editing,…



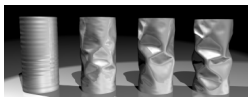
original | single resolution low pass | multiresolution low pass | mid stop band

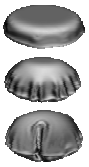


GEOMETRY PROCESSING

Stages

- creation, acquisition
- storage, transmission
- authentication
- editing, animation
- simulation
- manufacture





PARAMETERIZED SURFACES

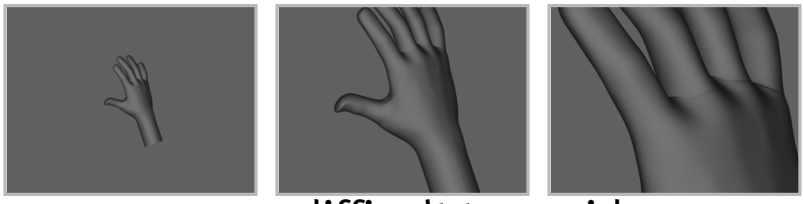
What about NURBS?

- spline patches are great, but…
  - difficult to stitch them together
    - trimming… (what a nightmare)
  - can be slow and cumbersome
  - we need scalable algorithms for the whole range from patches to fine meshes

Complex Shapes

Example: Building a hand

- Woody's hand from Pixar's Toy Story

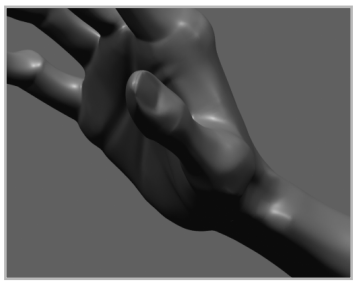


- very, very difficult to avoid seams

No More Seams

Subdivision solves the "stitching" problem

- a *single* smooth surface is defined
- example:
  - Geri's hand (Geri's Game; Pixar)



Patches

Advantages

- high level control (control points)
- compact representation
- multiresolution structure

Disadvantages

- difficult to maintain and manage
- naïve rendering of large models slow

Polygonal Meshes

Advantages

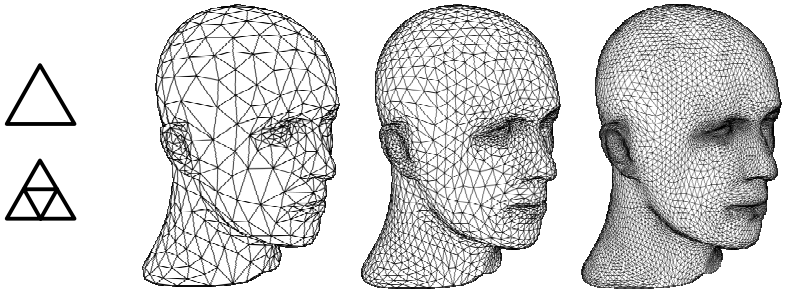
- very general
- direct hardware implementation

Disadvantages

- heavy weight representation
- good editing semantics difficult
- limited multiresolution structure

What is Subdivision?

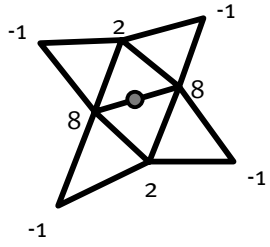
Define a smooth surface as the limit of a sequence of successive refinements

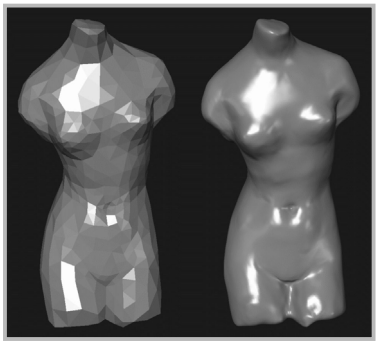


Interpolating

Keep old points, insert new ones

- affine combination of nearby points

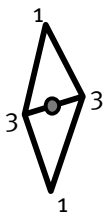


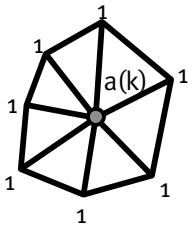


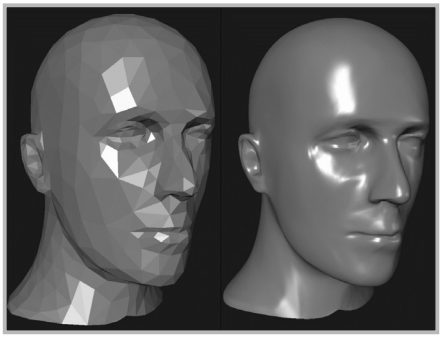
Approximating

Insert new, smooth new and old

- generalizes spline patches



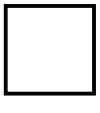


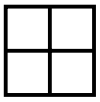


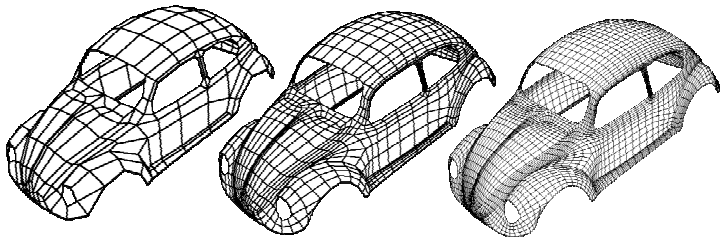
Lots of Flavors

Quadrilateral

- interpolating: Kobbelt scheme







Subdivision Surfaces

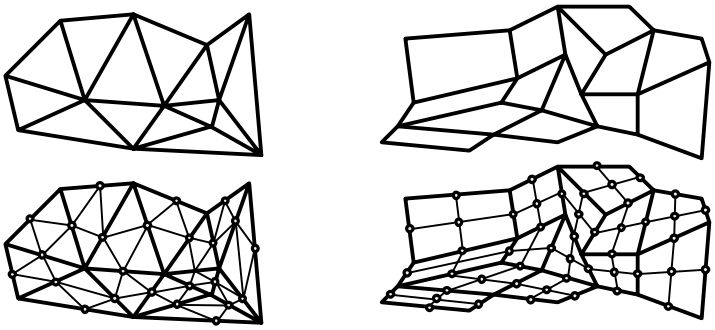
Important modeling primitive

- smooth, arbitrary topology surface modeling
- generalizes spline patches
- covers range of representations from "pure" patches to "pure" meshes
  - BUT: special connectivity (more on that later)

The Basic Setup (2D)

Topological rule

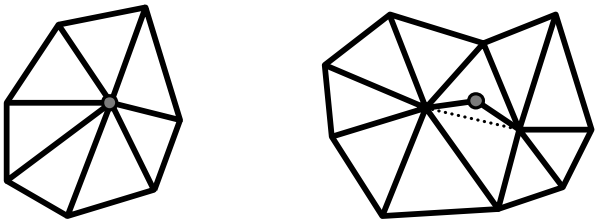
- modify connectivity



The Basic Setup (2D)

Geometric rule

- compute geometric positions
  - local linear combination of points



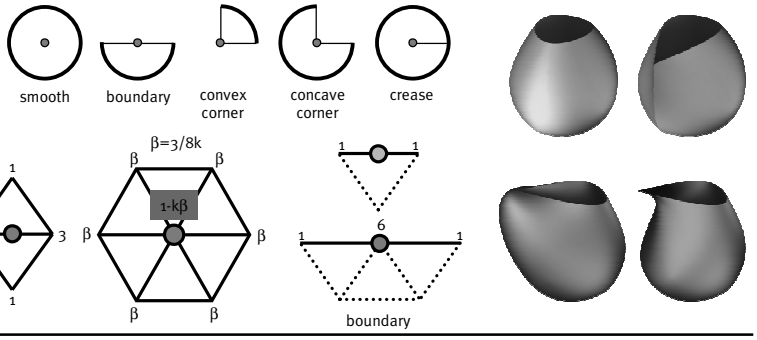
even at level i

odd at level i

Loop Scheme

Generalizes quartic box splines

- very simple rules

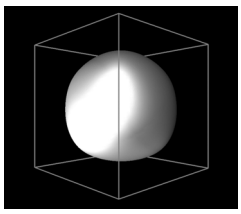


SUBDIVISION

Established schemes

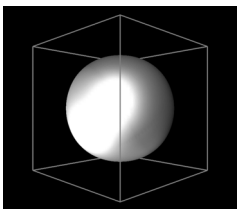
- Catmull-Clark
  - generalizes bi-cubic patches
- Loop
  - generalizes quartic box splines
- many others:
  - Doo-Sabin, Butterfly, Kobbelt, Peters/Reif

COMPARISON

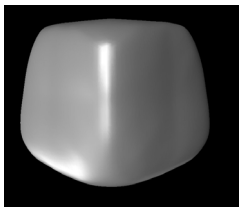


Loop

Catmull-Clark

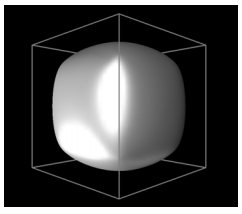


Cube

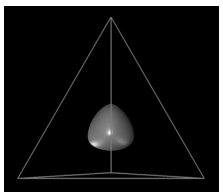


Butterfly

Doo-Sabin

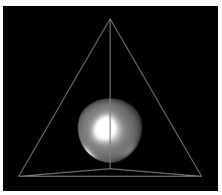


Comparison

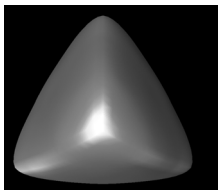


Loop

Catmull-Clark

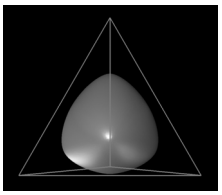


Tetrahedron

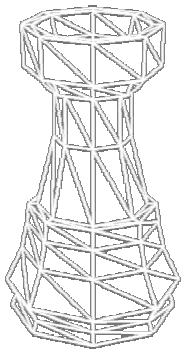


Butterfly

Doo-Sabin



Comparison

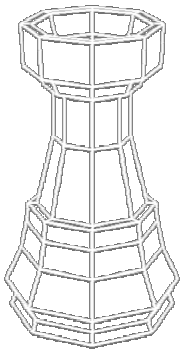


initial mesh

Loop

Catmull-Clark

Comparison

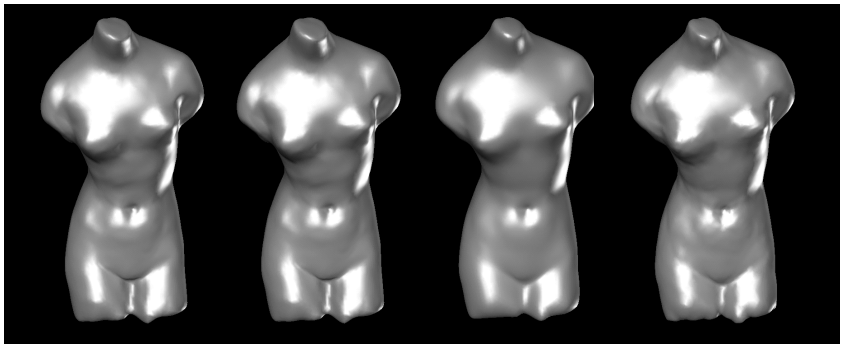


initial mesh

Loop

Catmull-Clark

Comparison



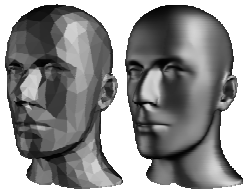
Butterfly | Catmull-Clark | Loop | Doo-Sabin

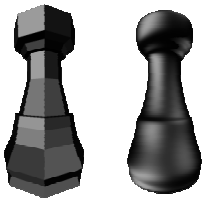
Why Subdivision

Many advantages

- arbitrary topology
- scalable
- wavelet connection
- easy to implement
- efficient

From meshes to surfaces!





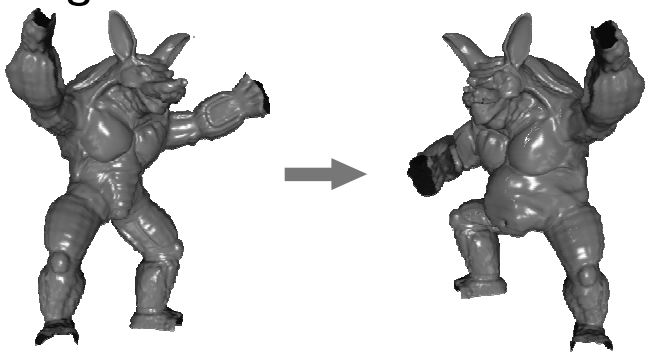
Multiresolution

What does it offer?

- good editing semantics
- deep connection with wavelets
  - compression, solvers, speed/accuracy tradeoff, approximation properties
- builtin support for LOD display
- very efficient

Application Example

Interactive multiresolution mesh editing



Multiresolution

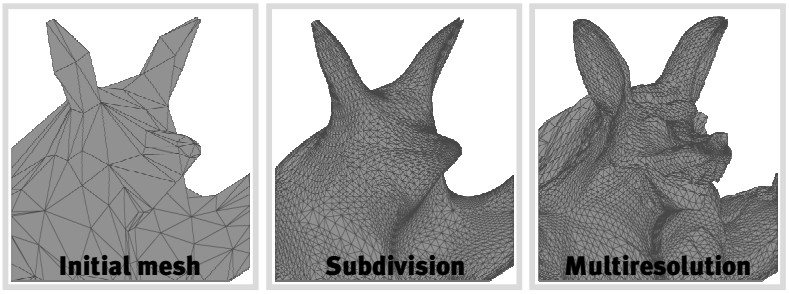
Subdivision alone not enough

- where are the details?
- just add!
  - parameterization of details important for editing
  - may be different for transmission
- pure wavelet setting not good enough

Multiresolution

Extension of subdivision

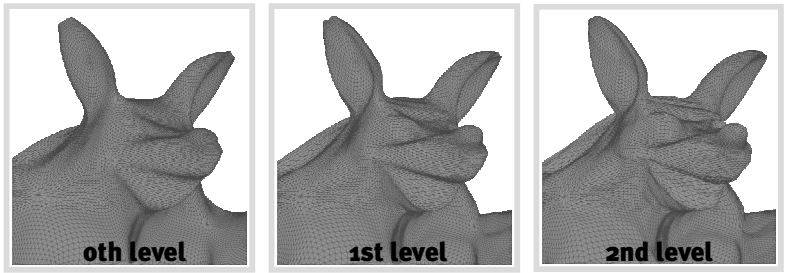
- details at different scales



Successive Scales

Add details in every subdivision step

- subdivide then displace

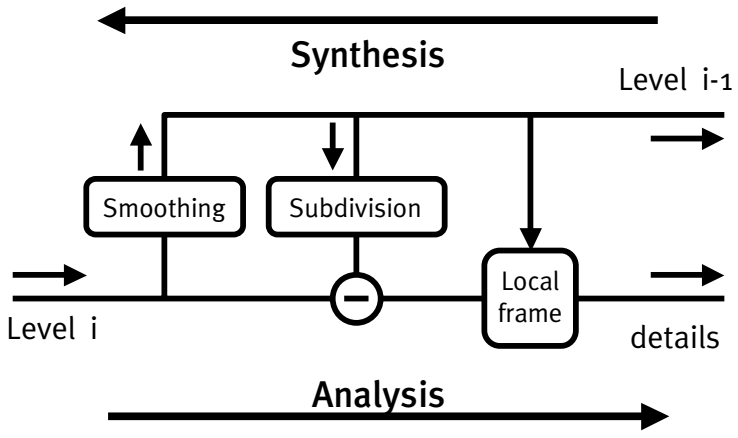


Algorithms

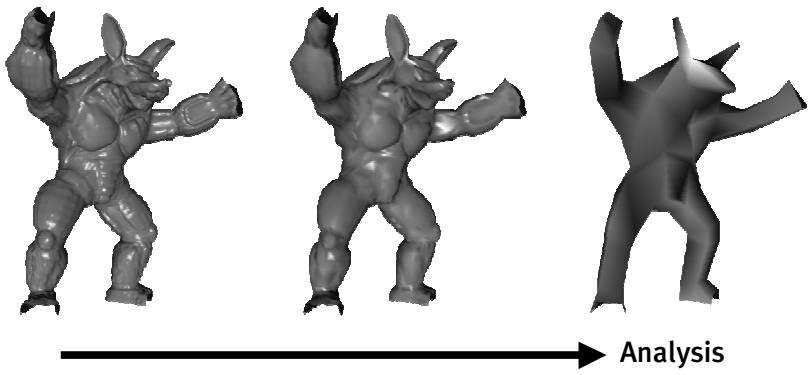
Two main components

- Synthesis: **coarse to fine**
  - subdivision
  - adding details (displacements)
- Analysis: **fine to coarse**
  - smoothing (Taubin)
  - computing details

Defining Details



Levels of Detail



From Fine to Coarse

Defining details

- apply Taubin smoother to fine level
- subdivide and take difference
- express in local coordinate frame induced by coarser level

Details

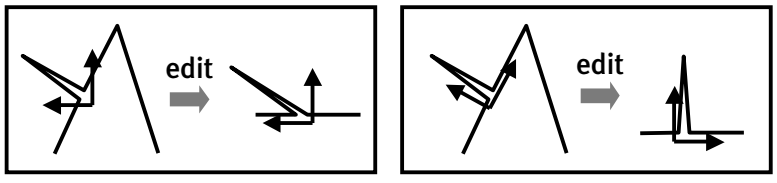
Correct editing behavior

- requires local frames
  - transform becomes non-linear
- no subsampling performed
  - Laplacian pyramid type construction
  - otherwise massive aliasing

Local vs Global Frame

Editing behavior

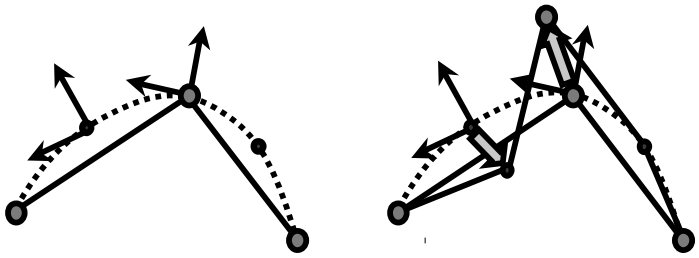
- global frame: simpler
- local frame: typically better



Global Local

Representing Details

Details represented in local frame



Adaptive Control

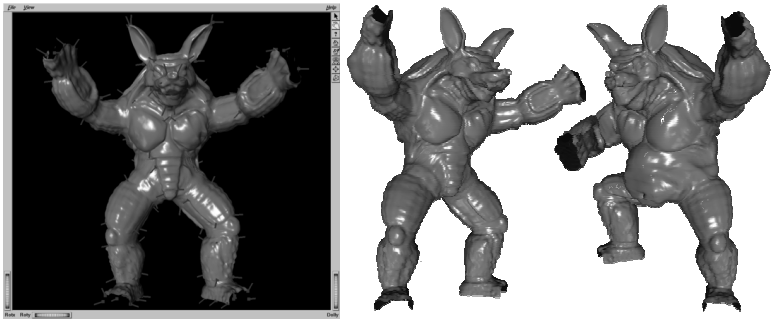
Thresholds

- adaptive analysis
  - ignore details which are too small
- adaptive synthesis
  - approximate given geometry sufficiently
- adaptive rendering
  - stay within hardware triangle budget

Example

Armadillo man

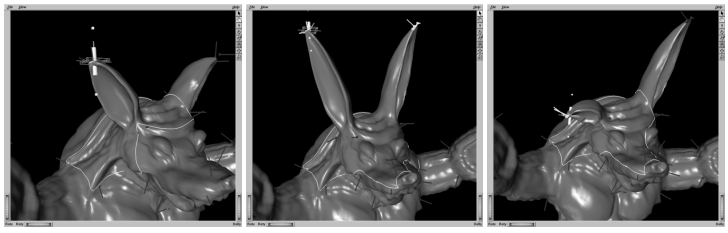
- 220k triangles of scanned geometry



Coarse Level Edit

Control vertices

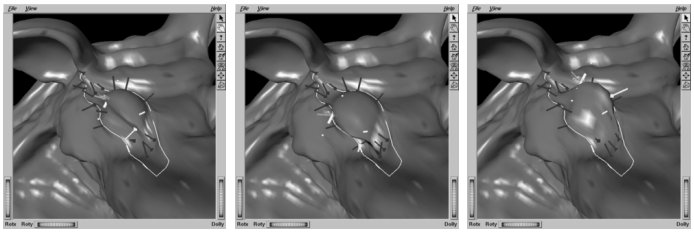
- pull group at coarsest level



Fine Level Edit

Control vertices

- descend several levels



Summary

Subdivision

- fundamental modeling paradigm
  - spans patches to meshes continuum
- simple algorithms
- attractive properties
- close connection with wavelets

SUMMARY

Multiresolution

- extend subdivision through addition of details
  - local coordinate frames important
- rendering bound application greatly accelerated
- details much better to code

Interactive Multiresolution Mesh Editing

Denis Zorin* Caltech
Peter Schröder† Caltech
Wim Sweldens‡ Bell Laboratories

Abstract

We describe a multiresolution representation for meshes based on subdivision, which is a natural extension of the existing patch-based surface representations. Combining subdivision and the smoothing algorithms of Taubin [26] allows us to construct a set of algorithms for interactive multiresolution editing of complex hierarchical meshes of arbitrary topology. The simplicity of the underlying algorithms for refinement and coarsification enables us to make them local and adaptive, thereby considerably improving their efficiency. We have built a scalable interactive multiresolution editing system based on such algorithms.

1 Introduction

Applications such as special effects and animation require creation and manipulation of complex geometric models of arbitrary topology. Like real world geometry, these models often carry detail at many scales (cf. Fig. 1). The model might be constructed from scratch (ab initio design) in an interactive modeling environment or be scanned-in either by hand or with automatic digitizing methods. The latter is a common source of data particularly in the entertainment industry. When using laser range scanners, for example, individual models are often composed of high resolution meshes with hundreds of thousands to millions of triangles.

Manipulating such fine meshes can be difficult, especially when they are to be edited or animated. Interactivity, which is crucial in these cases, is challenging to achieve. Even without accounting for any computation on the mesh itself, available rendering resources alone, may not be able to cope with the sheer size of the data. Possible approaches include mesh optimization [15, 13] to reduce the size of the meshes.

Aside from considerations of economy, the choice of representation is also guided by the need for multiresolution editing semantics. The representation of the mesh needs to provide control at a large scale, so that one can change the mesh in a broad, smooth manner, for example. Additionally designers will typically also want control over the minute features of the model (cf. Fig. 1). Smoother approximations can be built through the use of patches [14], though at the cost of loosing the high frequency details. Such detail can be reintroduced by combining patches with displacement maps [17]. However, this is difficult to manage in the arbitrary topology setting and across a continuous range of scales and hardware resources.

Figure 1: Before the Armadillo started working out he was flabby, complete with a double chin. Now he exercises regularly. The original is on the right (courtesy Venkat Krischnamurthy). The edited version on the left illustrates large scale edits, such as his belly, and smaller scale edits such as his double chin; all edits were performed at about 5 frames per second on an Indigo R10000 Solid Impact.

For reasons of efficiency the algorithms should be highly adaptive and dynamically adjust to available resources. Our goal is to have a single, simple, uniform representation with scalable algorithms. The system should be capable of delivering multiple frames per second update rates even on small workstations taking advantage of lower resolution representations.

In this paper we present a system which possesses these properties

- **Multiresolution control:** Both broad and general handles, as well as small knobs to tweak minute detail are available.
- **Speed/fidelity tradeoff:** All algorithms dynamically adapt to available resources to maintain interactivity.
- **Simplicity/uniformity:** A single primitive, triangular mesh, is used to represent the surface across all levels of resolution.

Our system is inspired by a number of earlier approaches. We mention multiresolution editing [11, 9, 12], arbitrary topology subdivision [6, 2, 19, 7, 28, 16], wavelet representations [21, 24, 8, 3], and mesh simplification [13, 17]. Independently an approach similar to ours was developed by Pulli and Lounsbery [23].

It should be noted that our methods rely on the finest level mesh having subdivision connectivity. This requires a remeshing step before external high resolution geometry can be imported into the editor. Eck et al. [8] have described a possible approach to remeshing arbitrary finest level input meshes fully automatically. A method that relies on a user's expertise was developed by Krishnamurthy and Levoy [17].

1.1 Earlier Editing Approaches

H-splines were presented in pioneering work on hierarchical editing by Forsey and Bartels [11]. Briefly, H-splines are obtained by adding finer resolution B-splines onto an existing coarser resolution B-spline patch relative to the coordinate frame induced by the

*dzorin@gg.caltech.edu
†ps@cs.caltech.edu
‡wim@bell-labs.com

coarser patch. Repeating this process, one can build very complicated shapes which are entirely parameterized over the unit square. Forsey and Bartels observed that the hierarchy induced coordinate frame for the offsets is essential to achieve correct editing semantics.

H-splines provide a uniform framework for representing both the coarse and fine level details. Note however, that as more detail is added to such a model the internal control mesh data structures more and more resemble a fine polyhedral mesh.

While their original implementation allowed only for regular topologies their approach could be extended to the general setting by using surface splines or one of the spline derived general topology subdivision schemes [18]. However, these schemes have not yet been made to work adaptively.

Forsey and Bartels' original work focused on the ab initio design setting. There the user's help is enlisted in defining what is meant by different levels of resolution. The user decides where to add detail and manipulates the corresponding controls. This way the levels of the hierarchy are hand built by a human user and the representation of the final object is a function of its editing history.

To edit an a priori given model it is crucial to have a general procedure to define coarser levels and compute details between levels. We refer to this as the *analysis* algorithm. An H-spline analysis algorithm based on weighted least squares was introduced [10], but is too expensive to run interactively. Note that even in an ab initio design setting online analysis is needed, since after a long sequence of editing steps the H-spline is likely to be overly refined and needs to be consolidated.

Wavelets provide a framework in which to rigorously define multiresolution approximations and fast analysis algorithms. Finkelstein and Salesin [9], for example, used B-spline wavelets to describe multiresolution editing of curves. As in H-splines, parameterization of details with respect to a coordinate frame induced by the coarser level approximation is required to get correct editing semantics. Gortler and Cohen [12], pointed out that wavelet representations of detail tend to behave in undesirable ways during editing and returned to a pure B-spline representation as used in H-splines.

Carrying these constructions over into the arbitrary topology surface framework is not straightforward. In the work by Lounsbery et al. [21] the connection between wavelets and subdivision was used to define the different levels of resolution. The original constructions were limited to piecewise linear subdivision, but smoother constructions are possible [24, 28].

An approach to surface modeling based on variational methods was proposed by Welch and Witkin [27]. An attractive characteristic of their method is flexibility in the choice of control points. However, they use a global optimization procedure to compute the surface which is not suitable for interactive manipulation of complex surfaces.

Before we proceed to a more detailed discussion of editing we first discuss different surface representations to motivate our choice of synthesis (refinement) algorithm.

1.2 Surface Representations

There are many possible choices for surface representations. Among the most popular are polynomial patches and polygons.

Patches are a powerful primitive for the construction of coarse grain, smooth models using a small number of control parameters. Combined with hardware support relatively fast implementations are possible. However, when building complex models with many patches the preservation of smoothness across patch boundaries can be quite cumbersome and expensive. These difficulties are compounded in the arbitrary topology setting when polynomial parameterizations cease to exist everywhere. Surface splines [4, 20, 22] provide one way to address the arbitrary topology challenge.

As more fine level detail is needed the proliferation of control points and patches can quickly overwhelm both the user and the most powerful hardware. With detail at finer levels, patches become less suited and polygonal meshes are more appropriate.

Polygonal Meshes can represent arbitrary topology and resolve fine detail as found in laser scanned models, for example. Given that most hardware rendering ultimately resolves to triangle scan-conversion even for patches, polygonal meshes are a very basic primitive. Because of sheer size, polygonal meshes are difficult to manipulate interactively. Mesh simplification algorithms [13] provide one possible answer. However, we need a mesh simplification approach, that is hierarchical and gives us shape handles for smooth changes over larger regions while maintaining high frequency details.

Patches and fine polygonal meshes represent two ends of a spectrum. Patches efficiently describe large smooth sections of a surface but cannot model fine detail very well. Polygonal meshes are good at describing very fine detail accurately using dense meshes, but do not provide coarser manipulation semantics.

Subdivision connects and unifies these two extremes.

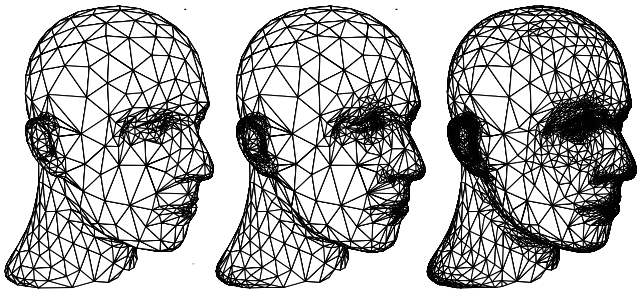


Figure 2: Subdivision describes a smooth surface as the limit of a sequence of refined polyhedra. The meshes show several levels of an adaptive Loop surface generated by our system (dataset courtesy Hugues Hoppe, University of Washington).

Subdivision defines a smooth surface as the limit of a sequence of successively refined polyhedral meshes (cf. Fig. 2). In the regular patch based setting, for example, this sequence can be defined through well known knot insertion algorithms [5]. Some subdivision methods generalize spline based knot insertion to irregular topology control meshes [2, 6, 19] while other subdivision schemes are independent of splines and include a number of interpolating schemes [7, 28, 16].

Since subdivision provides a path from patches to meshes, it can serve as a good foundation for the unified infrastructure that we seek. A single representation (hierarchical polyhedral meshes) supports the patch-type semantics of manipulation *and* finest level detail polyhedral edits equally well. The main challenge is to make the basic algorithms fast enough to escape the exponential time and space growth of naive subdivision. This is the core of our contribution.

We summarize the main features of subdivision important in our context

- **Topological Generality:** Vertices in a triangular (resp. quadrilateral) mesh need not have valence 6 (resp. 4). Generated surfaces are smooth everywhere, and efficient algorithms exist for computing normals and limit positions of points on the surface.
- **Multiresolution:** because they are the limit of successive refinement, subdivision surfaces support multiresolution algorithms, such as level-of-detail rendering, multiresolution editing, compression, wavelets, and numerical multigrid.

- **Simplicity:** subdivision algorithms are simple: the finer mesh is built through insertion of new vertices followed by *local* smoothing.
- **Uniformity of Representation:** subdivision provides a single representation of a surface at all resolution levels. Boundaries and features such as creases can be resolved through modified rules [14, 25], reducing the need for trim curves, for example.

1.3 Our Contribution

Aside from our perspective, which unifies the earlier approaches, our major contribution—and the main challenge in this program—is the design of highly adaptive and dynamic data structures and algorithms, which allow the system to function across a range of computational resources from PCs to workstations, delivering as much interactive fidelity as possible with a given polygon rendering performance. Our algorithms work for the class of 1-ring subdivision schemes (definition see below) and we demonstrate their performance for the concrete case of Loop's subdivision scheme.

The particulars of those algorithms will be given later, but Fig. 3 already gives a preview of how the different algorithms make up the editing system. In the next sections we first talk in more detail about subdivision, smoothing, and multiresolution transforms.

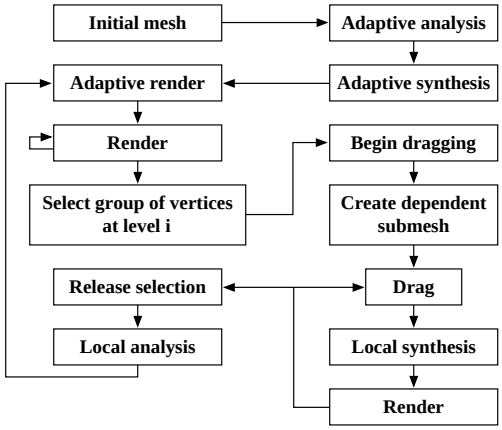


Figure 3: The relationship between various procedures as the user moves a set of vertices.

2 Subdivision

We begin by defining subdivision and fixing our notation. There are 2 points of view that we must distinguish. On the one hand we are dealing with an abstract *graph* and perform topological operations on it. On the other hand we have a *mesh* which is the geometric object in 3-space. The mesh is the image of a map defined on the graph: it associates a *point* in 3D with every *vertex* in the graph (cf. Fig. 4). A *triangle* denotes a face in the graph or the associated polygon in 3-space.

Initially we have a triangular graph T^0 with vertices V^0. By recursively *refining* each triangle into 4 subtriangles we can build a sequence of finer triangulations T^i with vertices V^i, $i > 0$ (cf. Fig. 4). The superscript i indicates the *level* of triangles and vertices respectively. A triangle $t \in T^i$ is a triple of indices $t = \{v_a, v_b, v_c\} \subset V^i$.

The vertex sets are nested as $V^j \subset V^i$ if $j < i$. We define *odd* vertices on level i as $M^i = V^{i+1} \setminus V^i$. V^{i+1} consists of two disjoint sets: *even* vertices (V^i) and *odd* vertices (M^i). We define the *level* of a vertex v as the smallest i for which $v \in V^i$. The level of v is $i+1$ if and only if $v \in M^i$.

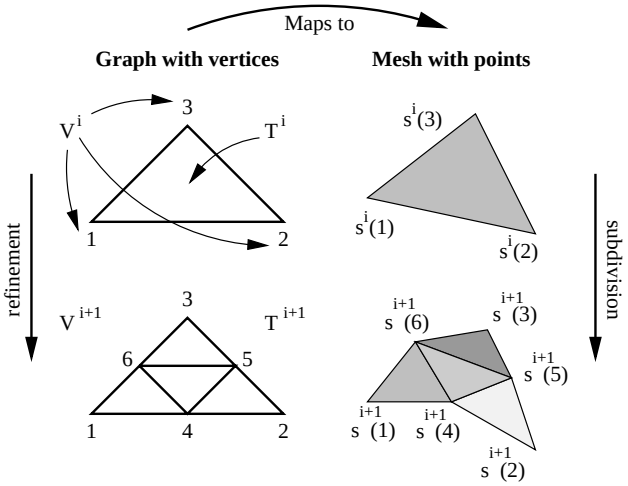


Figure 4: Left: the abstract graph. Vertices and triangles are members of sets V^i and T^i respectively. Their index indicates the level of refinement when they first appeared. Right: the mapping to the mesh and its subdivision in 3-space.

With each set V^i we associate a map, i.e., for each vertex v and each level i we have a 3D point $s^i(v) \in \mathbf{R}^3$. The set s^i contains all points on level i, $s^i = \{s^i(v) \mid v \in V^i\}$. Finally, a *subdivision scheme* is a linear operator S which takes the points from level i to points on the *finer* level $i+1$: $s^{i+1} = S\, s^i$

Assuming that the subdivision converges, we can define a limit surface σ as

$$\sigma = \lim_{k \to \infty} S^k\, s^0.$$

$\sigma(v) \in \mathbf{R}^3$ denotes the point on the limit surface associated with vertex v.

In order to define our offsets with respect to a local frame we also need tangent vectors and a normal. For the subdivision schemes that we use, such vectors can be defined through the application of linear operators Q and R acting on s^i so that $q^i(v) = (Qs^i)(v)$ and $r^i(v) = (Rs^i)(v)$ are linearly independent tangent vectors at $\sigma(v)$. Together with an orientation they define a local orthonormal frame $F^i(v) = (n^i(v), q^i(v), r^i(v))$. It is important to note that in general it is not necessary to use precise normals and tangents during editing; as long as the frame vectors are affinely related to the positions of vertices of the mesh, we can expect intuitive editing behavior.

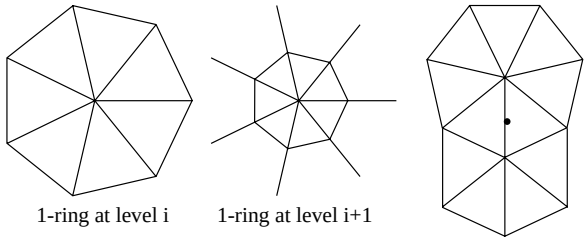


Figure 5: An even vertex has a 1-ring of neighbors at each level of refinement (left/middle). Odd vertices—in the middle of edges—have 1-rings around each of the vertices at either end of their edge (right).

Next we discuss two common subdivision schemes, both of which belong to the class of *1-ring schemes*. In these schemes points at level $i+1$ depend only on 1-ring neighborhoods of points

at level i. Let $v \in V^i$ (v even) then the point $s^{i+1}(v)$ is a function of only those $s^i(v_n)$, $v_n \in V^i$, which are immediate neighbors of v (cf. Fig. 5 left/middle). If $m \in M^i$ (m odd), it is the vertex inserted when splitting an edge of the graph; we call such vertices *middle vertices* of edges. In this case the point $s^{i+1}(m)$ is a function of the 1-rings around the vertices at the ends of the edge (cf. Fig. 5 right).

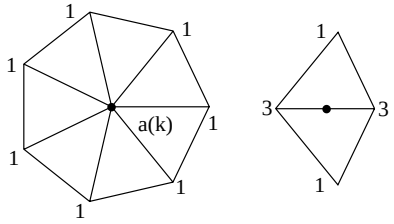


Figure 6: Stencils for Loop subdivision with unnormalized weights for even and odd vertices.

Loop is a non-interpolating subdivision scheme based on a generalization of quartic triangular box splines [19]. For a given even vertex $v \in V^i$, let $v_k \in V^i$ with $1 \le k \le K$ be its K 1-ring neighbors. The new point $s^{i+1}(v)$ is defined as $s^{i+1}(v) = (a(K)+K)^{-1}(a(K)\,s^i(v) + \sum_{k=1}^{K} s^i(v_k))$ (cf. Fig. 6), $a(K) = K(1-\alpha(K))/\alpha(K)$, and $\alpha(K) = 5/8-(3+2\cos(2\pi/K))^2/64$. For odd v the weights shown in Fig. 6 are used. Two independent tangent vectors $t_1(v)$ and $t_2(v)$ are given by $t_p(v) = \sum_{k=1}^{K} \cos(2\pi(k+p)/K)\, s^i(v_k)$.

Features such as boundaries and cusps can be accommodated through simple modifications of the stencil weights [14, 25, 29].

Butterfly is an interpolating scheme, first proposed by Dyn et al. [7] in the topologically regular setting and recently generalized to arbitrary topologies [28]. Since it is interpolating we have $s^i(v) = \sigma(v)$ for $v \in V^i$ even. The exact expressions for odd vertices depend on the valence K and the reader is referred to the original paper for the exact values [28].

For our implementation we have chosen the Loop scheme, since more performance optimizations are possible in it. However, the algorithms we discuss later work for any 1-ring scheme.

3 Multiresolution Transforms

So far we only discussed subdivision, i.e., how to go from coarse to fine meshes. In this section we describe analysis which goes from fine to coarse.

We first need *smoothing*, i.e., a linear operation H to build a smooth coarse mesh at level $i-1$ from a fine mesh at level i:

$$s^{i-1} = H\, s^i.$$

Several options are available here:

- **Least squares:** One could define analysis to be optimal in the least squares sense,

$$\min_{s^{i-1}} \|s^i - S\, s^{i-1}\|^2.$$

  The solution may have unwanted undulations and is too expensive to compute interactively [10].

- **Fairing:** A coarse surface could be obtained as the solution to a global variational problem. This is too expensive as well. An alternative is presented by Taubin [26], who uses a *local* non-shrinking smoothing approach.

Because of its computational simplicity we decided to use a version of Taubin smoothing. As before let $v \in V^i$ have K neighbors $v_k \in V^i$. Use the average, $\overline{s}^i(v) = K^{-1} \sum_{k=1}^{K} s^i(v_k)$, to define the discrete Laplacian $\mathcal{L}(v) = \overline{s}^i(v) - s^i(v)$. On this basis Taubin gives a Gaussian-like smoother which does not exhibit shrinkage

$$H := (I + \mu\,\mathcal{L})\,(I + \lambda\,\mathcal{L}).$$

With subdivision and smoothing in place, we can describe the transform needed to support multiresolution editing. Recall that for multiresolution editing we want the difference between successive levels expressed with respect to a frame induced by the coarser level, i.e., the offsets are relative to the smoother level.

With each vertex v and each level $i > 0$ we associate a *detail vector*, $d^i(v) \in \mathbf{R}^3$. The set d^i contains all detail vectors on level i, $d^i = \{d^i(v) \mid v \in V^i\}$. As indicated in Fig. 7 the detail vectors are defined as

$$d^i = (F^i)^t\,(s^i - S\,s^{i-1}) = (F^i)^t\,(I - S\,H)\,s^i,$$

i.e., the detail vectors at level i record how much the points at level i differ from the result of subdividing the points at level $i-1$. This difference is then represented with respect to the local frame F^i to obtain coordinate independence.

Since detail vectors are sampled on the fine level mesh V^i, this transformation yields an overrepresentation in the spirit of the Burt-Adelson Laplacian pyramid [1]. The only difference is that the smoothing filters (Taubin) are not the dual of the subdivision filter (Loop). Theoretically it would be possible to subsample the detail vectors and only record a detail per odd vertex of M^{i-1}. This is what happens in the wavelet transform. However, subsampling the details severely restricts the family of smoothing operators that can be used.

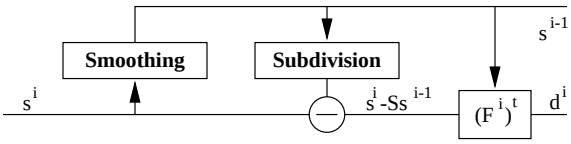


Figure 7: Wiring diagram of the multiresolution transform.

4 Algorithms and Implementation

Before we describe the algorithms in detail let us recall the overall structure of the mesh editor (cf. Fig 3). The analysis stage builds a succession of coarser approximations to the surface, each with fewer control parameters. Details or offsets between successive levels are also computed. In general, the coarser approximations are not visible; only their control points are rendered. These control points give rise to a *virtual surface* with respect to which the remaining details are given. Figure 8 shows wireframe representations of virtual surfaces corresponding to control points on levels 0, 1, and 2.

When an edit level is selected, the surface is represented internally as an approximation at this level, plus the set of all finer level details. The user can freely manipulate degrees of freedom at the edit level, while the finer level details remain unchanged relative to the coarser level. Meanwhile, the system will use the synthesis algorithm to render the modified edit level with all the finer details added in. In between edits, analysis enforces consistency on the internal representation of coarser levels and details (cf. Fig. 9).

The basic algorithms `Analysis` and `Synthesis` are very simple and we begin with their description.

Let $i = 0$ be the coarsest and $i = n$ the finest level with N vertices. For each vertex v and all levels i finer than the first level

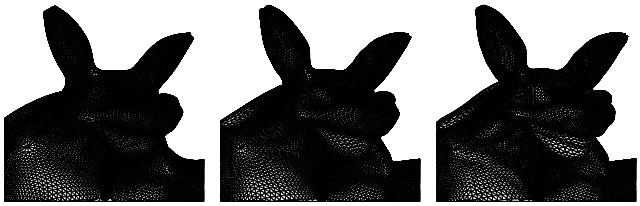


Figure 8: Wireframe renderings of virtual surfaces representing the first three levels of control points.

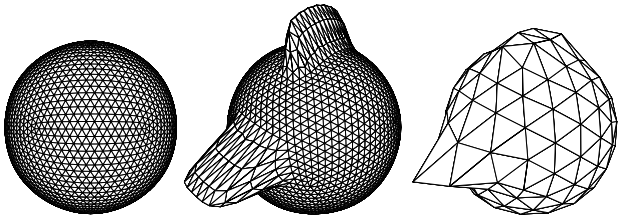


Figure 9: Analysis propagates the changes on finer levels to coarser levels, keeping the magnitude of details under control. Left: The initial mesh. Center: A simple edit on level 3. Right: The effect of the edit on level 2. A significant part of the change was absorbed by higher level details.

where the vertex v appears, there are storage locations $v.s[i]$ and $v.d[i]$, each with 3 floats. With this the total storage adds to $2 * 3 * (4N/3)$ floats. In general, $v.s[i]$ holds $s^i(v)$ and $v.d[i]$ holds $d^i(v)$; temporarily, these locations can be used to store other quantities. The local frame is computed by calling $v.F(i)$.

Global analysis and synthesis are performed level wise:

```
Analysis

  for i = n downto 1
    Analysis(i)
```

```
Synthesis

  for i = 1 to n
    Synthesis(i)
```

With the action at each level described by

```
Analysis(i)

  ∀v ∈ V^{i-1} : v.s[i-1] := smooth(v, i)
  ∀v ∈ V^i     : v.d[i] := v.F(i)^t * (v.s[i] - subd(v, i-1))
```

and

```
Synthesis(i)

  ∀v ∈ V^i : s.v[i] := v.F(i) * v.d[i] + subd(v, i-1)
```

Analysis computes points on the coarser level $i-1$ using smoothing (`smooth`), subdivides s^{i-1} (`subd`), and computes the detail vectors d^i (cf. Fig. 7). Synthesis reconstructs level i by subdividing level $i-1$ and adding the details.

So far we have assumed that all levels are uniformly refined, i.e., all neighbors at all levels exist. Since time and storage costs grow exponentially with the number of levels, this approach is unsuitable for an interactive implementation. In the next sections we explain how these basic algorithms can be made memory and time efficient.

Adaptive and *local* versions of these generic algorithms (cf. Fig. 3 for an overview of their use) are the key to these savings. The underlying idea is to use lazy evaluation and pruning based on thresholds. Three thresholds control this pruning: ϵ_A for adaptive analysis, ϵ_S for adaptive synthesis, and ϵ_R for adaptive rendering. To make lazy evaluation fast enough several caches are maintained explicitly and the order of computations is carefully staged to avoid recomputation.

4.1 Adaptive Analysis

The generic version of analysis traverses entire levels of the hierarchy starting at some finest level. Recall that the purpose of analysis is to compute coarser approximations and detail offsets. In many regions of a mesh, for example, if it is flat, no significant details will be found. *Adaptive analysis* avoids the storage cost associated with detail vectors below some threshold ϵ_A by observing that small detail vectors imply that the finer level almost coincides with the subdivided coarser level. The storage savings are realized through *tree pruning*.

For this purpose we need an integer $v.\mathit{finest} := \max_i\{\|v.d[i]\| \geq \epsilon_A\}$. Initially $v.\mathit{finest} = n$ and the following precondition holds before calling `Analysis(i)`:

- The surface is uniformly subdivided to level i,
- $\forall v \in V^i : v.s[i] = s^i(v)$,
- $\forall v \in V^i \mid i < j \leq v.\mathit{finest} : v.d[j] = d^j(v)$.

Now `Analysis(i)` becomes:

```
Analysis(i)

  ∀v ∈ V^{i-1} : v.s[i-1] := smooth(v, i)
  ∀v ∈ V^i :
    v.d[i] := v.s[i] - subd(v, i-1)
    if v.finest > i or ||v.d[i]|| ≥ ε_A then
      v.d[i] := v.F(i)^t * v.d[i]
    else
      v.finest := i - 1
  Prune(i - 1)
```

Triangles that do not contain details above the threshold are unrefined:

```
Prune(i)

  ∀t ∈ T^i : If all middle vertices m have m.finest = i - 1
             and all children are leaves, delete children.
```

This results in an adaptive mesh structure for the surface with $v.d[i] = d^i(v)$ for all $v \in V^i$, $i \leq v.\mathit{finest}$. Note that the resulting mesh is not restricted, i.e., two triangles that share a vertex can differ in more than one level. Initial analysis has to be followed by a synthesis pass which enforces restriction.

4.2 Adaptive Synthesis

The main purpose of the general synthesis algorithm is to rebuild the finest level of a mesh from its hierarchical representation. Just as in the case of analysis we can get savings from noticing that in flat regions, for example, little is gained from synthesis and one might as well save the time and storage associated with synthesis. This is the basic idea behind *adaptive synthesis*, which has two main purposes. First, ensure the mesh is restricted on each level, (cf. Fig. 10). Second, refine triangles and recompute points until the mesh has reached a certain measure of local flatness compared against the threshold ϵ_S.

The algorithm recomputes the points $s^i(v)$ starting from the coarsest level. Not all neighbors needed in the subdivision stencil of a given point necessarily exist. Consequently adaptive synthesis

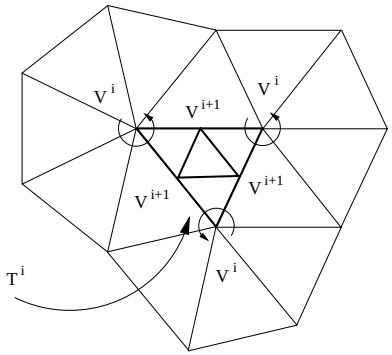


Figure 10: A restricted mesh: the center triangle is in T^i and its vertices in V^i. To subdivide it we need the 1-rings indicated by the circular arrows. If these are present the graph is restricted and we can compute s^{i+1} for all vertices and middle vertices of the center triangle.

```
Refine(t, i, dir)

  if t.leaf then  Create children for t
  ∀v ∈ t : if v.depth < i + 1 then
    GetRing(v, i)
    Update(v, i)
    ∀m ∈ N(v, i + 1, 1) :
      Update(m, i)
      if m.finest ≥ i + 1 then
        forced := true
  if dir and Flat(t) < ε_S and not forced then
    Delete children of t
  else
    ∀t ∈ current : t.restrict := true

Update(v, i)
  v.s[i + 1] := subd(v, i)
  v.depth := i + 1
  if v.finest ≥ i + 1 then
    v.s[i + 1] += v.F(i + 1) * v.d[i + 1]
```

lazily creates all triangles needed for subdivision by temporarily refining their parents, then computes subdivision, and finally deletes the newly created triangles unless they are needed to satisfy the restriction criterion. The following precondition holds before entering `AdaptiveSynthesis`:

- $\forall t \in T^j \mid 0 \le j \le i$: t is restricted
- $\forall v \in V^j \mid 0 \le j \le v.depth$: $v.s[j] = s^j(v)$

where $v.depth := \max_i\{s^i(v)$has been recomputed$\}$.

```
AdaptiveSynthesis

  ∀v ∈ V^0 : v.depth := 0
  for i = 0 to n − 1
    temptri := {}
    ∀t ∈ T^i :
      current := {}
      Refine(t, i, true)
    ∀t ∈ temptri : if not t.restrict then
      Delete children of t
```

The list *temptri* serves as a cache holding triangles from levels $j < i$ which are temporarily refined. A triangle is appended to the list if it was refined to compute a value at a vertex. After processing level i these triangles are unrefined unless their $t.restrict$ flag is set, indicating that a temporarily created triangle was later found to be needed permanently to ensure restriction. Since triangles are appended to *temptri*, parents precede children. Deallocating the list tail first guarantees that all unnecessary triangles are erased.

The function `Refine`(t, i, dir) (see below) creates children of $t \in T^i$ and computes the values $Ss^i(v)$ for the vertices and middle vertices of t. The results are stored in $v.s[i + 1]$. The boolean argument *dir* indicates whether the call was made directly or recursively.

The condition $v.depth = i + 1$ indicates whether an earlier call to `Refine` already recomputed $s^{i+1}(v)$. If not, call `GetRing`(v, i) and `Update`(v, i) to do so. In case a detail vector lives at v at level i ($v.finest \ge i + 1$) add it in. Next compute $s^{i+1}(m)$ for middle vertices on level $i + 1$ around v ($m \in N(v, i + 1, 1)$, where $N(v, i, l)$ is the l-ring neighborhood of vertex v at level i). If m has to be calculated, compute `subd`(m, i) and add in the detail if it exists and record this fact in the flag *forced* which will prevent unrefinement later. At this point, all s^{i+1} have been recomputed for the vertices and middle vertices of t. Unrefine t and delete its children if `Refine` was called directly, the triangle is sufficiently flat, and none of the middle vertices contain details (i.e., $forced =$ `false`). The list *current* functions as a cache holding triangles from level $i - 1$ which are temporarily refined to build a 1-ring around the vertices of t. If after processing all vertices and middle vertices of t it is decided that t will remain refined, none of the coarser-level triangles from *current* can be unrefined without violating restriction. Thus $t.restrict$ is set for all of them. The function `Flat`(t) measures how close to planar the corners and edge middle vertices of t are.

Finally, `GetRing`(v, i) ensures that a complete ring of triangles on level i adjacent to the vertex v exists. Because triangles on level i are restricted triangles all triangles on level $i - 1$ that contain v exist (precondition). At least one of them is refined, since otherwise there would be no reason to call `GetRing`(v, i). All other triangles could be leaves or temporarily refined. Any triangle that was already temporarily refined may become permanently refined to enforce restriction. Record such candidates in the *current* cache for fast access later.

```
GetRing(v, i)

  ∀t ∈ T^{i−1} with v ∈ t :
    if t.leaf then
      Refine(t, i − 1, false); temptri.append(t)
      t.restrict := false; t.temp := true
    if t.temp then
      current.append(t)
```

4.3 Local Synthesis

Even though the above algorithms are adaptive, they are still run everywhere. During an edit, however, not all of the surface changes. The most significant economy can be gained from performing analysis and synthesis only over submeshes which require it.

Assume the user edits level l and modifies the points $s^l(v)$ for $v \in V^{*l} \subset V^l$. This invalidates coarser level values s^i and d^i for certain subsets $V^{*i} \subset V^i$, $i \leq l$, and finer level points s^i for subsets $V^{*i} \subset V^i$ for $i > l$. Finer level detail vectors d^i for $i > l$ remain correct by definition. Recomputing the coarser levels is done by *local incremental analysis* described in Section 4.4, recomputing the finer level is done by *local synthesis* described in this section.

The set of vertices V^{*i} which are affected depends on the support of the subdivision scheme. If the support fits into an m-ring around the computed vertex, then all modified vertices on level $i+1$ can be found recursively as

$$V^{*i+1} = \bigcup_{v \in V^{*i}} N(v, i+1, m).$$

We assume that $m = 2$ (Loop-like schemes) or $m = 3$ (Butterfly type schemes). We define the *subtriangulation* T^{*i} to be the subset of triangles of T^i with vertices in V^{*i}.

`LocalSynthesis` is only slightly modified from `AdaptiveSynthesis`: iteration starts at level l and iterates only over the submesh T^{*i}.

4.4 Local Incremental Analysis

After an edit on level l *local incremental analysis* will recompute $s^i(v)$ and $d^i(v)$ locally for coarser level vertices ($i \leq l$) which are affected by the edit. As in the previous section, we assume that the user edited a set of vertices v on level l and call V^{*i} the set of vertices affected on level i. For a given vertex $v \in V^{*i}$ we define

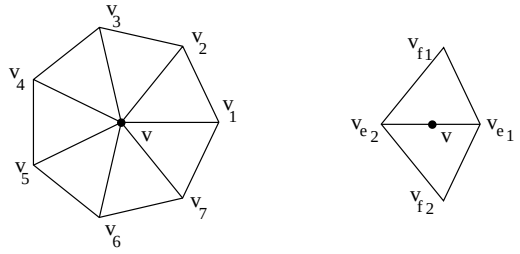


Figure 11: Sets of even vertices affected through smoothing by either an even v or odd m vertex.

$R^{i-1}(v) \subset V^{i-1}$ to be the set of vertices on level $i-1$ affected by v through the smoothing operator H. The sets V^{*i} can now be defined recursively starting from level $i = l$ to $i = 0$:

$$V^{*i-1} = \bigcup_{v \in V^{*i}} R^{i-1}(v).$$

The set $R^{i-1}(v)$ depends on the size of the smoothing stencil and whether v is even or odd (cf. Fig. 11). If the smoothing filter is 1-ring, e.g., Gaussian, then $R^{i-1}(v) = \{v\}$ if v is even and $R^{i-1}(m) = \{v_{e1}, v_{e2}\}$ if m is odd. If the smoothing filter is 2-ring, e.g., Taubin, then $R^{i-1}(v) = \{v\} \cup \{v_k \mid 1 \leq k \leq K\}$ if v is even and $R^{i-1}(m) = \{v_{e1}, v_{e2}, v_{f1}, v_{f2}\}$ if v is odd. Because of restriction, these vertices always exist. For $v \in V^i$ and $v' \in R^{i-1}(v)$ we let $c(v, v')$ be the coefficient in the analysis stencil. Thus

$$(H\, s^i)(v') = \sum_{v \mid v' \in R^{i-1}(v)} c(v, v') s^i(v).$$

This could be implemented by running over the v' and each time computing the above sum. Instead we use the dual implementation, iterate over all v, accumulating (`+=`) the right amount to $s^i(v')$ for $v' \in R^{i-1}(v)$. In case of a 2-ring Taubin smoother the coefficients are given by

$$\begin{aligned} c(v,v) &= (1-\mu)(1-\lambda) + \mu\lambda/6 \\ c(v,v_k) &= \mu\,\lambda/6K \\ c(m,v_{e1}) &= ((1-\mu)\lambda + (1-\lambda)\mu + \mu\lambda/3)/K \\ c(m,v_{f1}) &= \mu\,\lambda/3K, \end{aligned}$$

where for each $c(v, v')$, K is the outdegree of v'.

The algorithm first copies the old points $s^i(v)$ for $v \in V^{*i}$ and $i \leq l$ into the storage location for the detail. If then propagates the incremental changes of the modified points from level l to the coarser levels and adds them to the old points (saved in the detail locations) to find the new points. Then it recomputes the detail vectors that depend on the modified points.

We assume that before the edit, the old points $s^l(v)$ for $v \in V^{*l}$ were saved in the detail locations. The algorithm starts out by building V^{*i-1} and saving the points $s^{i-1}(v)$ for $v \in V^{*i-1}$ in the detail locations. Then the changes resulting from the edit are propagated to level $i-1$. Finally $S\, s^{i-1}$ is computed and used to update the detail vectors on level i.

```
LocalAnalysis(i)

  ∀v ∈ V^{*i} : ∀v' ∈ R^{i-1}(v) :
    V^{*i-1} ∪= {v'}
    v'.d[i-1] := v'.s[i-1]
  ∀v ∈ V^{*i} : ∀v' ∈ R^{i-1}(v) :
    v'.s[i-1] += c(v,v') * (v.s[i] - v.d[i])
  ∀v ∈ V^{*i-1} :
    v.d[i] = v.F(i)^t * (v.s[i] - subd(v, i-1))
    ∀m ∈ N(v,i,1) :
      m.d[i] = m.F(i)^t * (m.s[i] - subd(m, i-1))
```

Note that the odd points are actually computed twice. For the Loop scheme this is less expensive than trying to compute a predicate to avoid this. For Butterfly type schemes this is not true and one can avoid double computation by imposing an ordering on the triangles. The top level code is straightforward:

```
LocalAnalysis

  ∀v ∈ V^{*l} : v.d[l] := v.s[l]
  for i := l downto 0
    LocalAnalysis(i)
```

It is difficult to make incremental local analysis adaptive, as it is formulated purely in terms of vertices. It is, however, possible to adaptively clean up the triangles affected by the edit and (un)refine them if needed.

4.5 Adaptive Rendering

The *adaptive rendering* algorithm decides which triangles will be drawn depending on the rendering performance available and level of detail needed.

The algorithm uses a flag $t.draw$ which is initialized to `false`, but set to `true` as soon as the area corresponding to t is drawn. This can happen either when t itself gets drawn, or when a set of its descendents, which cover t, is drawn. The top level algorithm loops through the triangles starting from the level $n-1$. A triangle

is always responsible for drawing its children, never itself, unless it is a coarsest-level triangle.

```
AdaptiveRender

  for i = n − 1 downto 0
    ∀t ∈ T^i : if not t.leaf then
      Render(t)
  ∀t ∈ T^0 : if not t.draw then
    displaylist.append(t)
```

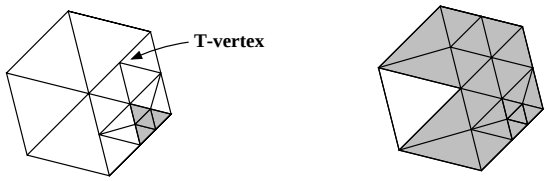


Figure 12: Adaptive rendering: On the left 6 triangles from level i, one has a covered child from level $i+1$, and one has a T-vertex. On the right the result from applying `Render` to all six.

The `Render`(t) routine decides whether the children of t have to be drawn or not (cf. Fig.12). It uses a function `edist`(m) which measures the distance between the point corresponding to the edge's middle vertex m, and the edge itself. In the when case any of the children of t are already drawn or any of its middle vertices are far enough from the plane of the triangle, the routine will draw the rest of the children and set the draw flag for all their vertices and t. It also might be necessary to draw a triangle if some of its middle vertices are drawn because the triangle on the other side decided to draw its children. To avoid cracks, the routine `cut`(t) will cut t into 2, 3, or 4, triangles depending on how many middle vertices are drawn.

```
Render(t)

  if (∃ c ∈ t.child | c.draw = true
   or ∃ m ∈ t.mid_vertex | edist(m) > ε_D) then
    ∀c ∈ t.child :
      if not c.draw then
        displaylist.append(c)
        ∀v ∈ c : v.draw := true
    t.draw := true
  else if ∃ m ∈ t.mid_vertex | m.draw = true
    ∀t' ∈ cut(t) : displaylist.append(t')
    t.draw := true
```

4.6 Data Structures and Code

The main data structure in our implementation is a forest of triangular quadtrees. Neighborhood relations within a single quadtree can be resolved in the standard way by ascending the tree to the least common parent when attempting to find the neighbor across a given edge. Neighbor relations between adjacent trees are resolved explicitly at the level of a collection of roots, i.e., triangles of a coarsest level graph. This structure also maintains an explicit representation of the boundary (if any). Submeshes rooted at any level can be created on the fly by assembling a new graph with some set of triangles as roots of their child quadtrees. It is here that the explicit representation of the boundary comes in, since the actual trees are never copied, and a boundary is needed to delineate the actual submesh.

The algorithms we have described above make heavy use of container classes. Efficient support for sets is essential for a fast implementation and we have used the C++ Standard Template Library. The mesh editor was implemented using OpenInventor and OpenGL and currently runs on both SGI and Intel PentiumPro workstations.

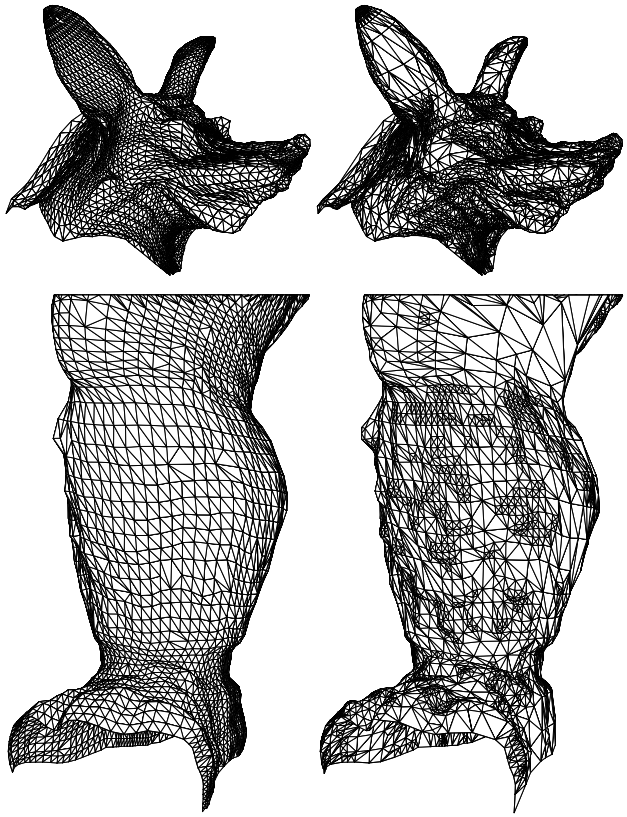


Figure 13: On the left are two meshes which are uniformly subdivided and consist of 11k (upper) and 9k (lower) triangles. On the right another pair of meshes mesh with approximately the same numbers of triangles. Upper and lower pairs of meshes are generated from the same original data but the right meshes were optimized through suitable choice of ϵ_S. See the color plates for a comparison between the two under shading.

5 Results

In this section we show some example images to demonstrate various features of our system and give performance measures.

Figure 13 shows two triangle mesh approximations of the Armadillo head and leg. Approximately the same number of triangles are used for both adaptive and uniform meshes. The meshes on the left were rendered uniformly, the meshes on the right were rendered adaptively. (See also color plate 15.)

Locally changing threshold parameters can be used to resolve an area of interest particularly well, while leaving the rest of the mesh at a coarse level. An example of this "lens" effect is demonstrated in Figure 14 around the right eye of the Mannequin head. (See also color plate 16.)

We have measured the performance of our code on two platforms: an Indigo R10000@175MHz with Solid Impact graphics, and a PentiumPro@200MHz with an Intergraph Intense 3D board.

We used the Armadillo head as a test case. It has approximately 172000 triangles on 6 levels of subdivision. Display list creation took 2 seconds on the SGI and 3 seconds on the PC for the full model. We adjusted ϵ_R so that both machines rendered models at 5 frames per second. In the case of the SGI approximately 113,000 triangles were rendered at that rate. On the PC we achieved 5 frames per second when the rendering threshold had been raised enough so that an approximation consisting of 35000 polygons was used.

The other important performance number is the time it takes to recompute and re-render the region of the mesh which is changing as the user moves a set of control points. This submesh is rendered in immediate mode, while the rest of the surface continues to be rendered as a display list. Grabbing a submesh of 20-30 faces (a typical case) at level 0 added 250 mS of time per redraw, at level 1 it added 110 mS and at level 2 it added 30 mS in case of the SGI. The corresponding timings for the PC were 500 mS, 200 mS and 60 mS respectively.

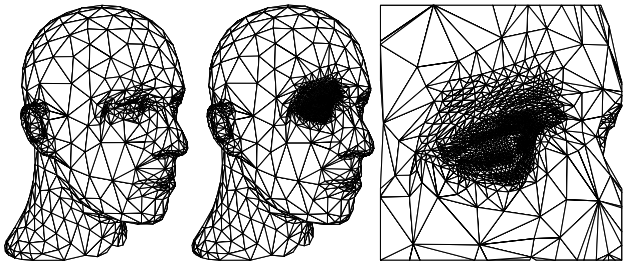


Figure 14: It is easy to change ϵ_S locally. Here a "lens" was applied to the right eye of the Mannequin head with decreasing ϵ_S to force very fine resolution of the mesh around the eye.

6 Conclusion and Future Research

We have built a scalable system for interactive multiresolution editing of arbitrary topology meshes. The user can either start from scratch or from a given fine detail mesh *with subdivision connectivity*. We use smooth subdivision combined with details at each level as a uniform surface representation across scales and argue that this forms a natural connection between fine polygonal meshes and patches. Interactivity is obtained by building both local and adaptive variants of the basic analysis, synthesis, and rendering algorithms, which rely on fast lazy evaluation and tree pruning. The system allows interactive manipulation of meshes according to the polygon performance of the workstation or PC used.

There are several avenues for future research:

- Multiresolution transforms readily connect with compression. We want to be able to store the models in a compressed format and use progressive transmission.
- Features such as creases, corners, and tension controls can easily be added into our system and expand the users' editing toolbox.
- Presently no real time fairing techniques, which lead to more intuitive coarse levels, exist.
- In our system coarse level edits can only be made by dragging coarse level vertices. Which vertices live on coarse levels is currently fixed because of subdivision connectivity. Ideally the user should be able to dynamically adjust this to make coarse level edits centered at arbitrary locations.
- The system allows topological edits on the coarsest level. Algorithms that allow topological edits on all levels are needed.
- An important area of research relevant for this work is generation of meshes with subdivision connectivity from scanned data or from existing models in other representations.

Acknowledgments

We would like to thank Venkat Krishnamurthy for providing the Armadillo dataset. Andrei Khodakovsky and Gary Wu helped beyond the call of duty to bring the system up. The research was supported in part through grants from the Intel Corporation, Microsoft, the Charles Lee Powell Foundation, the Sloan Foundation, an NSF CAREER award (ASC-9624957), and under a MURI (AFOSR F49620-96-1-0471). Other support was provided by the NSF STC for Computer Graphics and Scientific Visualization.

References

[1] BURT, P. J., AND ADELSON, E. H. Laplacian Pyramid as a Compact Image Code. *IEEE Trans. Commun. 31*, 4 (1983), 532–540.

[2] CATMULL, E., AND CLARK, J. Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. *Computer Aided Design 10*, 6 (1978), 350–355.

[3] CERTAIN, A., POPOVIĆ, J., DEROSE, T., DUCHAMP, T., SALESIN, D., AND STUETZLE, W. Interactive Multiresolution Surface Viewing. In *SIGGRAPH 96 Conference Proceedings*, H. Rushmeier, Ed., Annual Conference Series, 91–98, Aug. 1996.

[4] DAHMEN, W., MICCHELLI, C. A., AND SEIDEL, H.-P. Blossoming Begets B-Splines Bases Built Better by B-Patches. *Mathematics of Computation 59*, 199 (July 1992), 97–115.

[5] DE BOOR, C. *A Practical Guide to Splines*. Springer, 1978.

[6] DOO, D., AND SABIN, M. Analysis of the Behaviour of Recursive Division Surfaces near Extraordinary Points. *Computer Aided Design 10*, 6 (1978), 356–360.

[7] DYN, N., LEVIN, D., AND GREGORY, J. A. A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control. *ACM Trans. Gr. 9*, 2 (April 1990), 160–169.

[8] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. In *Computer Graphics Proceedings*, Annual Conference Series, 173–182, 1995.

[9] FINKELSTEIN, A., AND SALESIN, D. H. Multiresolution Curves. *Computer Graphics* Proceedings, Annual Conference Series, 261–268, July 1994.

[10] FORSEY, D., AND WONG, D. Multiresolution Surface Reconstruction for Hierarchical B-splines. Tech. rep., University of British Columbia, 1995.

[11] FORSEY, D. R., AND BARTELS, R. H. Hierarchical B-Spline Refinement. *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, No. 4, pp. 205–212, August 1988.

[12] GORTLER, S. J., AND COHEN, M. F. Hierarchical and Variational Geometric Modeling with Wavelets. In *Proceedings Symposium on Interactive 3D Graphics*, May 1995.

[13] HOPPE, H. Progressive Meshes. In *SIGGRAPH 96 Conference Proceedings*, H. Rushmeier, Ed., Annual Conference Series, 99–108, August 1996.

[14] HOPPE, H., DEROSE, T., DUCHAMP, T., HALSTEAD, M., JIN, H., MCDONALD, J., SCHWEITZER, J., AND STUETZLE, W. Piecewise Smooth Surface Reconstruction. In *Computer Graphics Proceedings*, Annual Conference Series, 295–302, 1994.

[15] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. Mesh Optimization. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, J. T. Kajiya, Ed., vol. 27, 19–26, August 1993.

[16] KOBBELT, L. Interpolatory Subdivision on Open Quadrilateral Nets with Arbitrary Topology. In *Proceedings of Eurographics 96*, Computer Graphics Forum, 409–420, 1996.

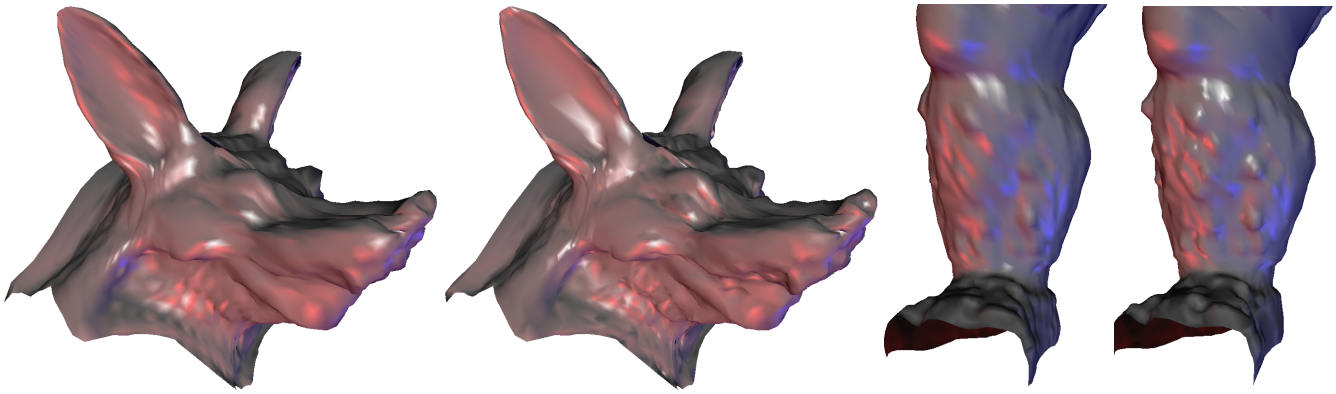


Figure 15: Shaded rendering (OpenGL) of the meshes in Figure 13.

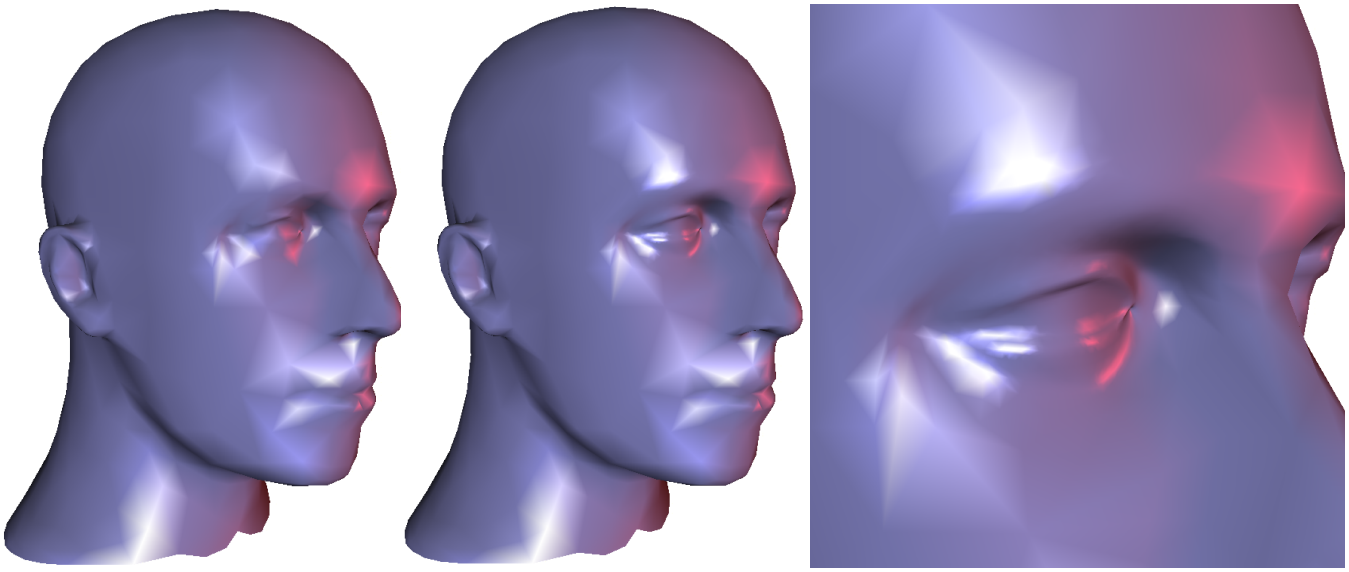


Figure 16: Shaded rendering (OpenGL) of the meshes in Figure 14.

[17] KRISHNAMURTHY, V., AND LEVOY, M. Fitting Smooth Surfaces to Dense Polygon Meshes. In *SIGGRAPH 96 Conference Proceedings*, H. Rushmeier, Ed., Annual Conference Series, 313–324, August 1996.

[18] KURIHARA, T. Interactive Surface Design Using Recursive Subdivision. In *Proceedings of Communicating with Virtual Worlds*. Springer Verlag, June 1993.

[19] LOOP, C. Smooth Subdivision Surfaces Based on Triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.

[20] LOOP, C. Smooth Spline Surfaces over Irregular Meshes. In *Computer Graphics Proceedings*, Annual Conference Series, 303–310, 1994.

[21] LOUNSBERY, M., DEROSE, T., AND WARREN, J. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *Transactions on Graphics 16*, 1 (January 1997), 34–73.

[22] PETERS, J. C^1 Surface Splines. *SIAM J. Numer. Anal. 32*, 2 (1995), 645–666.

[23] PULLI, K., AND LOUNSBERY, M. Hierarchical Editing and Rendering of Subdivision Surfaces. Tech. Rep. UW-CSE-97-04-07, Dept. of CS&E, University of Washington, Seattle, WA, 1997.

[24] SCHRÖDER, P., AND SWELDENS, W. Spherical wavelets: Efficiently representing functions on the sphere. *Computer Graphics Proceedings, (SIGGRAPH 95)* (1995), 161–172.

[25] SCHWEITZER, J. E. *Analysis and Application of Subdivision Surfaces*. PhD thesis, University of Washington, 1996.

[26] TAUBIN, G. A Signal Processing Approach to Fair Surface Design. In *SIGGRAPH 95 Conference Proceedings*, R. Cook, Ed., Annual Conference Series, 351–358, August 1995.

[27] WELCH, W., AND WITKIN, A. Variational surface modeling. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, E. E. Catmull, Ed., vol. 26, 157–166, July 1992.

[28] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interpolating Subdivision for Meshes with Arbitrary Topology. *Computer Graphics Proceedings (SIGGRAPH 96)* (1996), 189–192.

[29] ZORIN, D. N. *Subdivision and Multiresolution Surface Representations*. PhD thesis, Caltech, Pasadena, California, 1997.

Digital Geometry Processing

Denis Zorin, NYU

Jianbo Peng, Lexing Ying,

Henning Biermann, Aaron Hertzmann

new york university mrl

Problems

Idea: transfer image processing techniques to geometry

- two examples: denoising and texture synthesis
- problems: lack of parameterization, global direction fields, rectangular sampling pattern

new york university mrl

Denoising

Problem formulation:

Given

- signal contaminated with noise
- signal and noise model

Find

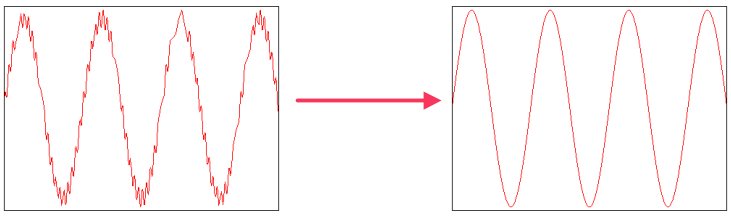
- best possible approximation to the original signal

new york university mrl

Models

Simple example:

- signal low-frequency
- noise high-frequency
- ideal denoising = low-pass filtering



new york university mrl

Image denoising

Transform to a "good" domain

e.g. wavelet thresholding

- wavelet transform
- eliminate small coefficients
- inverse wavelet transform

Problem

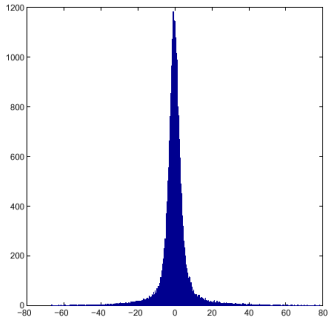
- assumes any large coefficient useful

new york university mrl

Approach

Statistical model for wavelet coefficients

- peak at zero
- heavy tails
- strong spatial magnitude correlation
- correlation across scales



new york university mrl

GSM model

Gaussian scale mixture

- General: vector X of coefficients near a point

$$X = \sqrt{z}U$$

- z is (random) scalar factor
- U is Gaussian vector

new york university mrl

Denoising

Simplest case

- no local correlation: $x = \sqrt{z}u$
- $x/\sqrt{z}$ is Gaussian
- noisy signal: $y = \sqrt{z}u + w$
- Wiener filtering - best we can do if noise is Gaussian

$$\hat{x} = \frac{z\sigma_u^2}{z\sigma_u^2 + \sigma_w^2} y$$

new york university mrl

Estimating scale

Need to know noise and scale

- distribution

$$P(x) = \int \frac{1}{(2\pi)^{1/2}\sqrt{z}\sigma_u} exp\left(-\frac{x^2}{2z\sigma_u^2}\right) P_z(z)dz$$

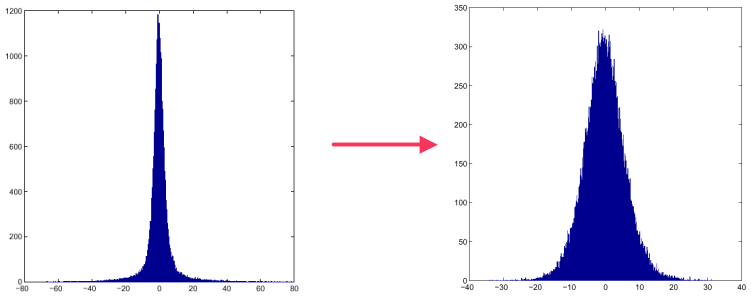
- max. likelihood estimate from a vector Y of coeffs on **neighborhood** of size N **:**

$$z\sigma_u^2 = \frac{Y^T Y}{N} - \sigma_w^2$$

new york university mrl

Rescaling

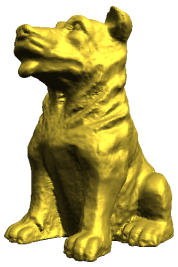
Divide by estimated scale



new york university mrl

Moving to geometry

Need to generalize

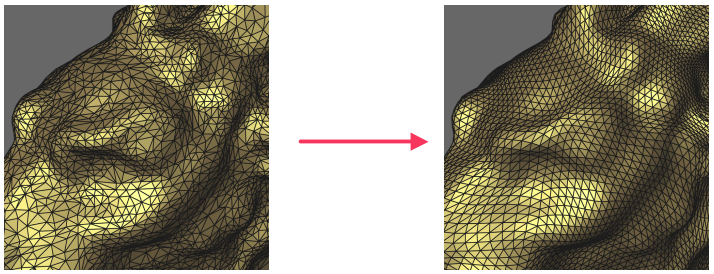


- multiscale decomposition
  - observation: does not have to be a basis
- directional filters
  - observation: do not need global orientation, only consistency in each neighborhood
- Everything else works

new york university mrl

Semiregular meshes

Assumption: reparameterize first

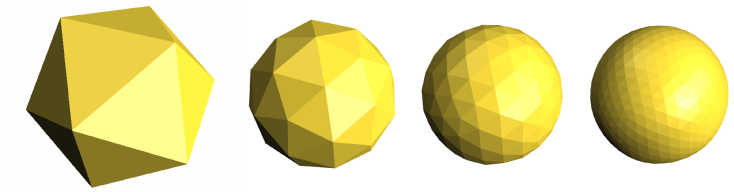


new york university mrl

Hierarchy

Subdivision connectivity

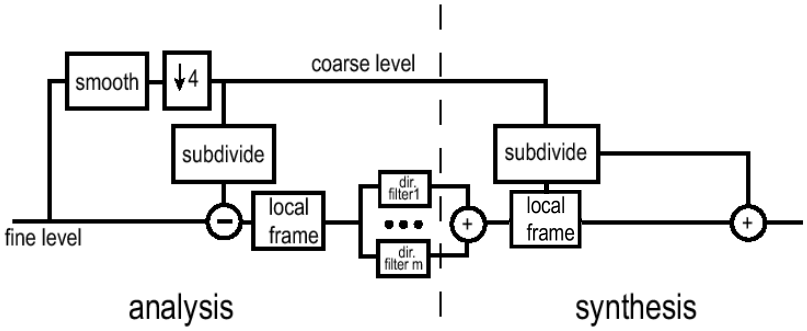
- can define multiscale decompositions



new york university mrl

Multiscale decomposition

Overrepresentation

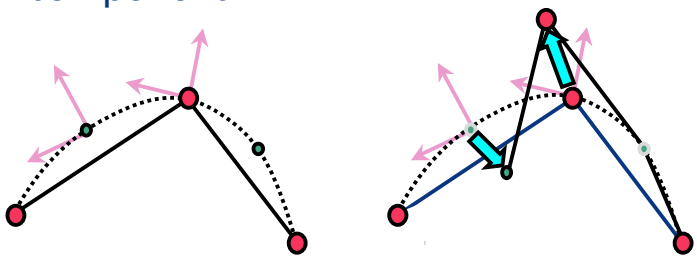


new york university mrl

Local frame

Computed from the coarser level

- more natural representation
- separate tangent from normal component

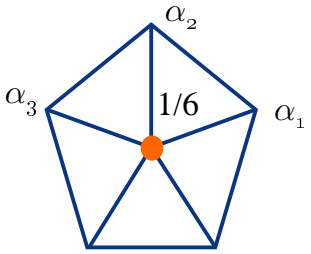


new york university mrl

Directional filters

Use 6 directions

- most natural for semiregular



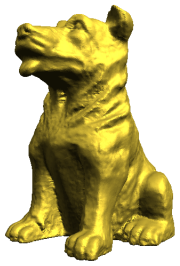
m-th filter:

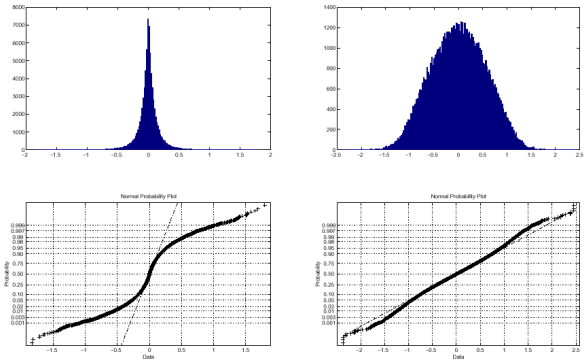
$$\alpha_i = \cos\left(\frac{2i\pi}{k} - \theta_m\right)$$

new york university mrl

Distributions

Similar to images





before dividing by $\sqrt{\hat{z}}$

after dividing by $\sqrt{\hat{z}}$

new york university mrl

Algorithm overview

Same basic steps as for images

Given estimated noise

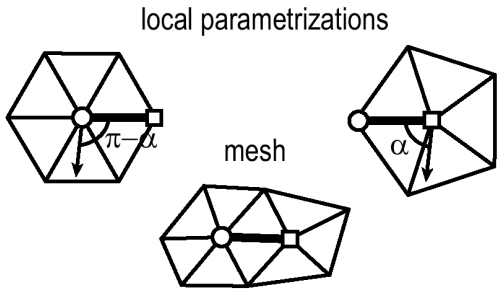
- transform to multiscale representation
- for each directional coefficient for every scale:
  - Estimate the scale factor z using immediate neighbors
  - apply Wiener-like filtering
- transform back

new york university mrl

Local alignment

No global bands

- when estimating z for a given direction, apply aligned filters to neighbors:



new york university mrl

How it works

Correlation taken into account through scale:

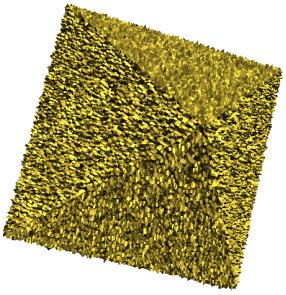
$$z\sigma_u^2 = \frac{Y^T Y}{N} - \sigma_w^2$$

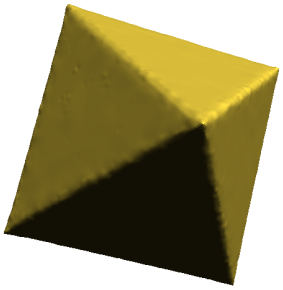
- if scale large, no attenuation
- if small, scale is zero and coefficient is eliminated

new york university mrl

Results

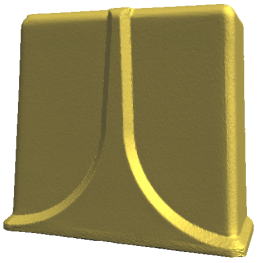
Artificial data, “12%” noise

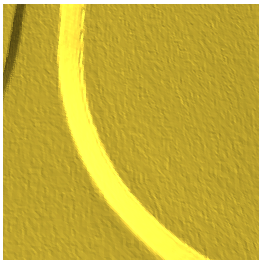




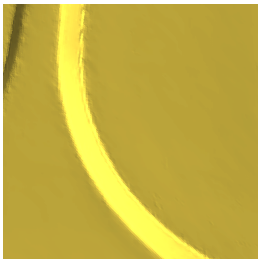
new york university mrl

Results





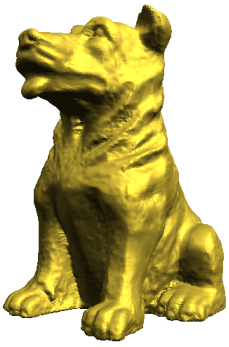
before

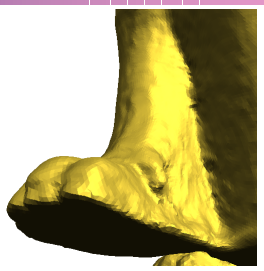


after

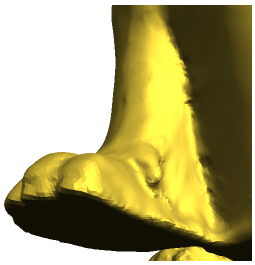
new york university mrl

Results

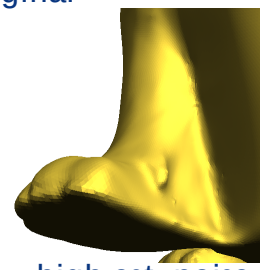




original



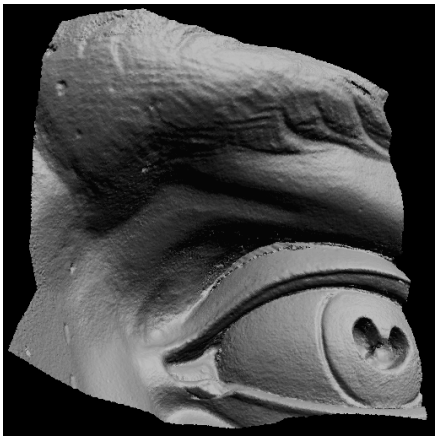
low est. noise



high est. noise

new york university mrl

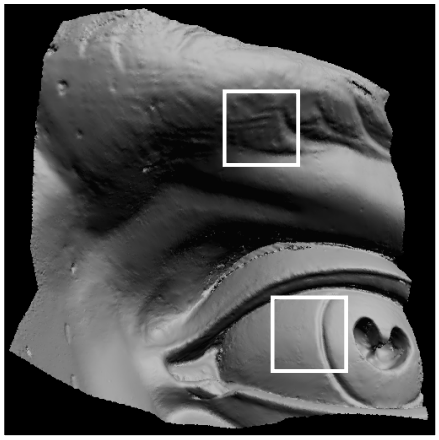
Results



Fragment of Michelangelo's David, 0.29 mm resolution

new york university mrl

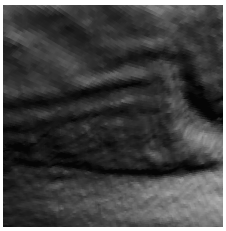
Results

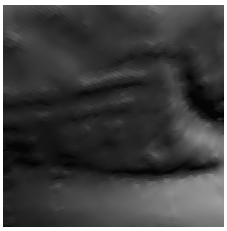


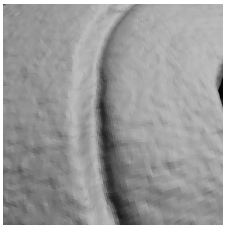
Fragment of Michelangelo's David, 0.29 mm resolution

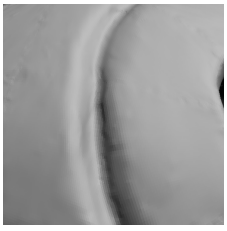
new york university mrl

Results









original denoised

new york university mrl

Curvature diffusion

Deterministic approach

- evolve surface along estimated normal at rate proportional to curvature
- to use for denoising, add anisotropic diffusion tensor
- tensor components depend on estimated principal curvatures (e.g. big ratio is indication of an edge)
- need to solve a PDE

new york university mrl

Comparison

Difficult to compare; similar results for suitable parameters

new york university mrl

Open questions

Only a first step

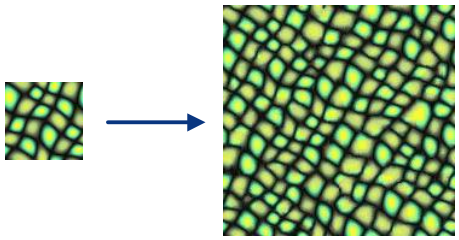
- sampling
- validation of the model
- interscale coherence
- integrate with compression

new york university mrl

Texture synthesis

Much recent work

- DeBonet (1997), Efros and Leung (1999), Wei and Levoy (2000), Ashikhmin (2001)



A **B**

Texture synthesis

Goal: synthsesis on surfaces directly

new york university mrl

Texture model

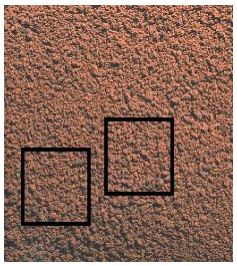
Li-Yi Wei, 2000

Textures are

- local
- stationary

Model textures by

- local spatial neighborhoods



new york university mrl

Basic algorithm

Exhaustive search

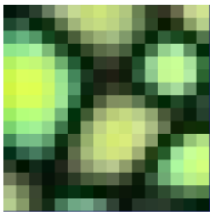
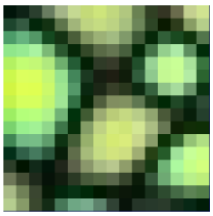


image by Li-Yi Wei, Stanford University

new york university mrl

Basic algorithm

Exhaustive search



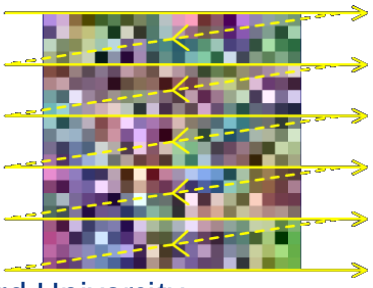


image by Li-Yi Wei, Stanford University

new york university mrl

Basic algorithm

Exhaustive search

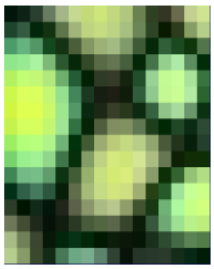


image by Li-Yi Wei, Stanford University

new york university mrl

Basic algorithm

Exhaustive search

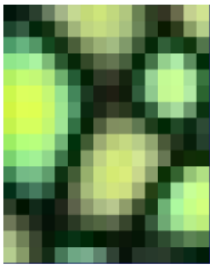


image by Li-Yi Wei, Stanford University

new york university mrl

Basic algorithm

Exhaustive search

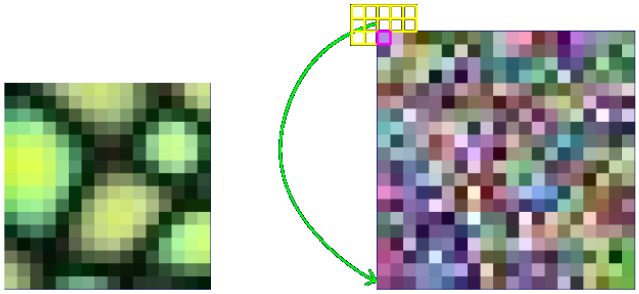


image by Li-Yi Wei, Stanford University

new york university mrl

Basic algorithm

Exhaustive search

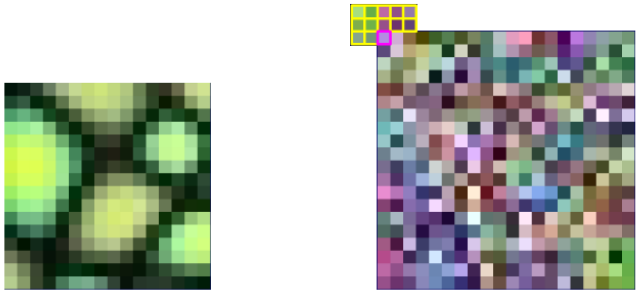


image by Li-Yi Wei, Stanford University

new york university mrl

Basic algorithm

Exhaustive search

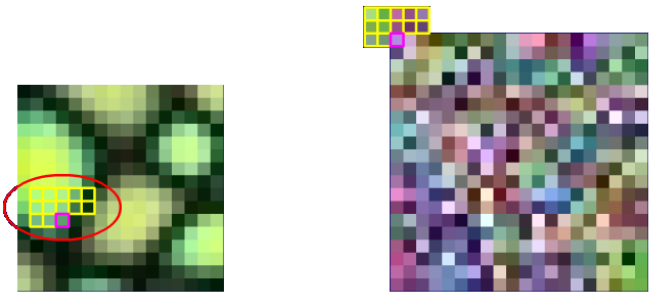


image by Li-Yi Wei, Stanford University

new york university mrl

Basic algorithm

Exhaustive search

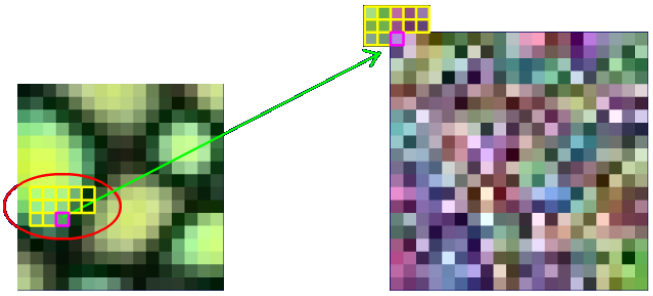
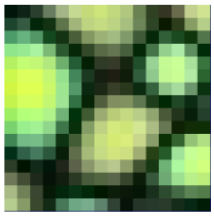


image by Li-Yi Wei, Stanford University

new york university mrl

Basic algorithm

Exhaustive search



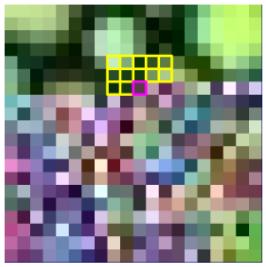
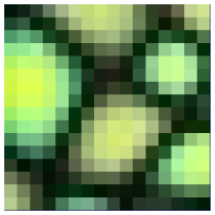


image by Li-Yi Wei, Stanford University

new york university mrl

Basic algorithm

Exhaustive search



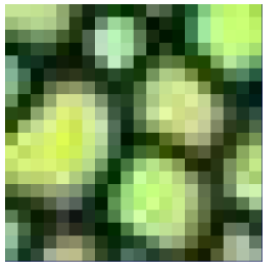


image by Li-Yi Wei, Stanford University

new york university mrl

Multiscale synthesis

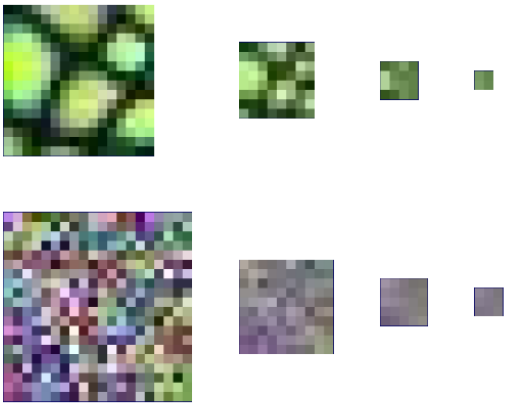


image by Li-Yi Wei, Stanford University

new york university mrl

Multiscale synthesis

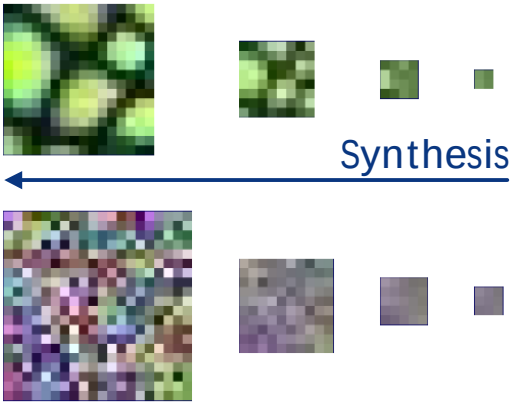


image by Li-Yi Wei, Stanford University

new york university mrl

Multiscale synthesis

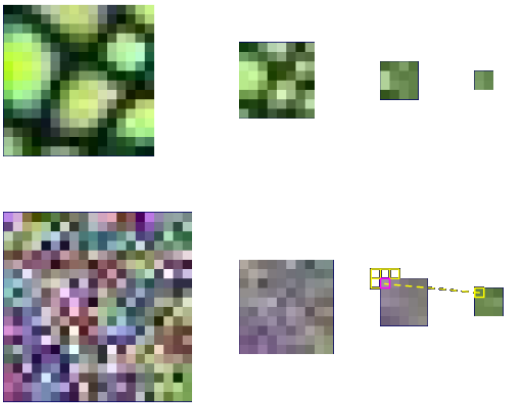
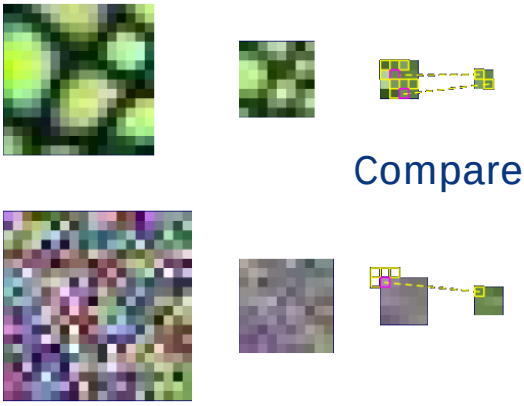


image by Li-Yi Wei, Stanford University

new york university mrl

Multiscale synthesis



Compare

image by Li-Yi Wei, Stanford University

new york university mrl

Multiscale synthesis

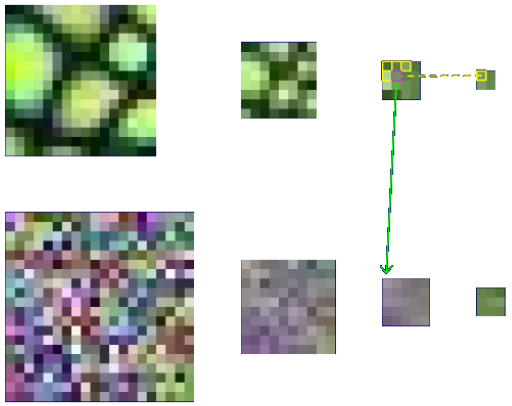


image by Li-Yi Wei, Stanford University

new york university mrl

Multiscale synthesis

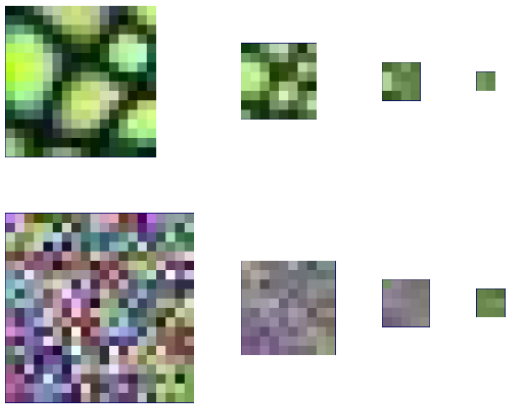


image by Li-Yi Wei, Stanford University

new york university mrl

Multiscale synthesis

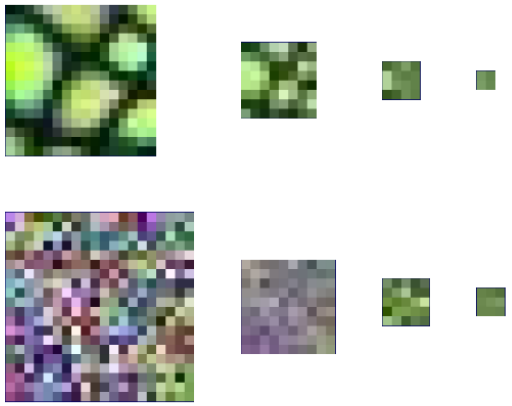


image by Li-Yi Wei, Stanford University

new york university mrl

Multiscale synthesis

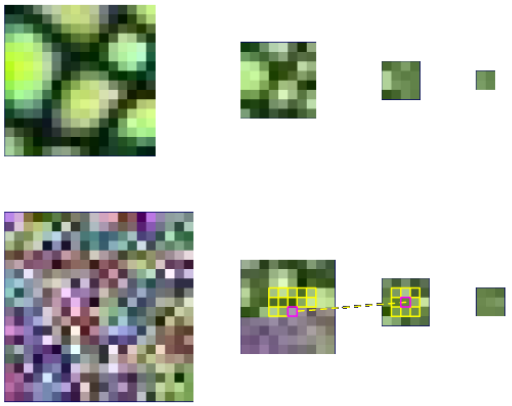


image by Li-Yi Wei, Stanford University

new york university mrl

Multiscale synthesis

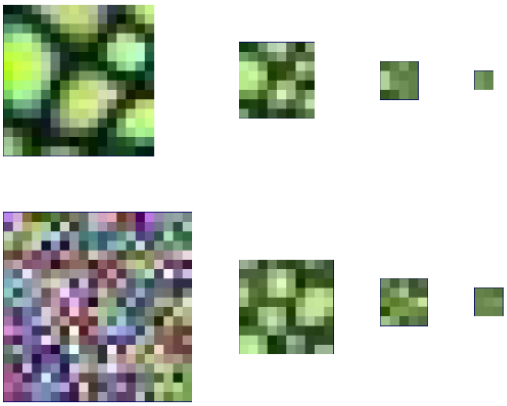


image by Li-Yi Wei, Stanford University

new york university mrl

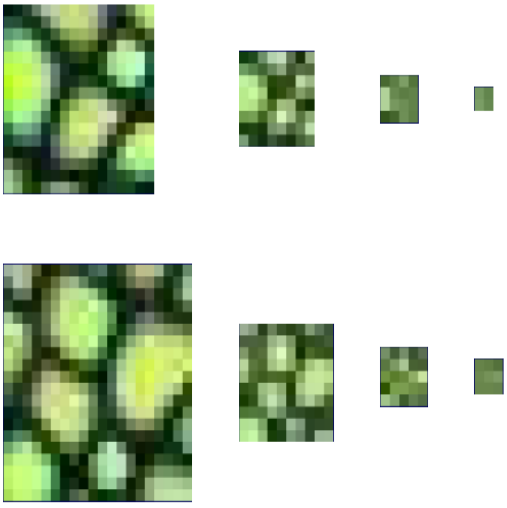


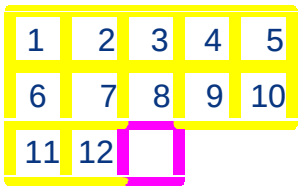
image by Li-Yi Wei, Stanford University

new york university mrl

Acceleration

Bottleneck: search for best match

- best (so far) solution: TSVQ
- IMPORTANT:
  requires identical sampling pattern for
  all neighborhoods



Neighborhood

new york university mrl

Generalizing to surfaces

Problems

- local sampling patterns may be different
- sampling density may vary
- anisotropic sampling
- traversal order undefined

Solution

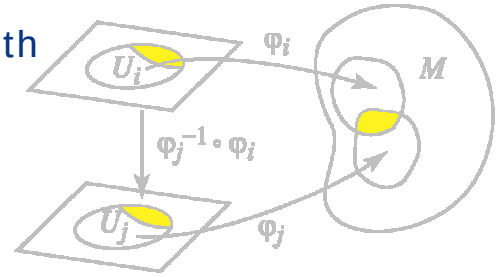
- treat texture as continuous function
- resample as necessary

new york university mrl

Charts

Atlas for the surface

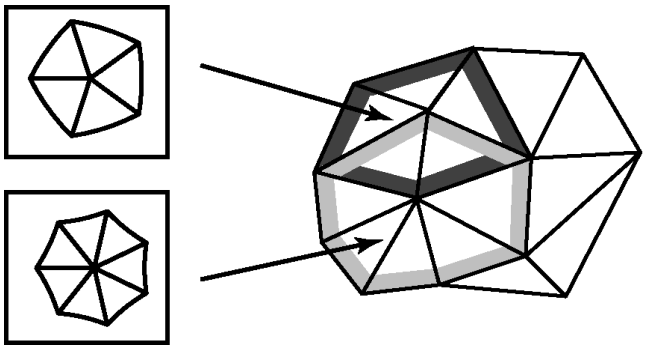
- every point is covered
- charts large enough for neighborhoods to fit inside
- should be smooth



new york university mrl

Subdivision charts

Use characteristic maps



new york university mrl

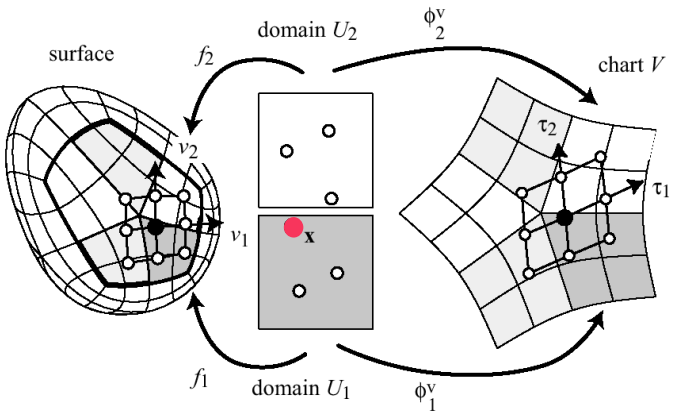
Chart resampling

Correct for distortion

- assume sampling pattern around x is small enough in world space
- compute the linear approximation A to the chart map at x
- sample in the chart space using a pattern which is mapped by A to the one we need

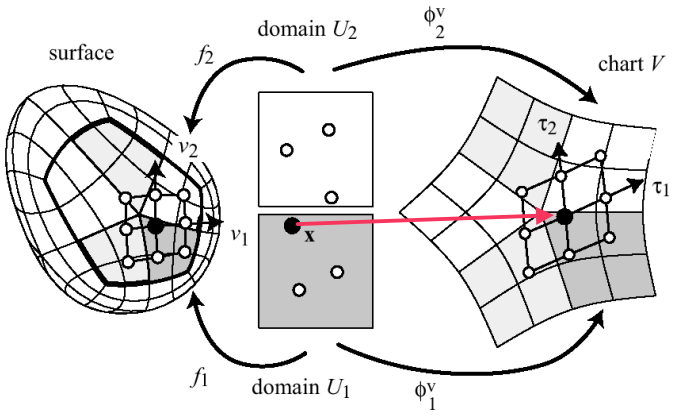
new york university mrl

Chart sampling



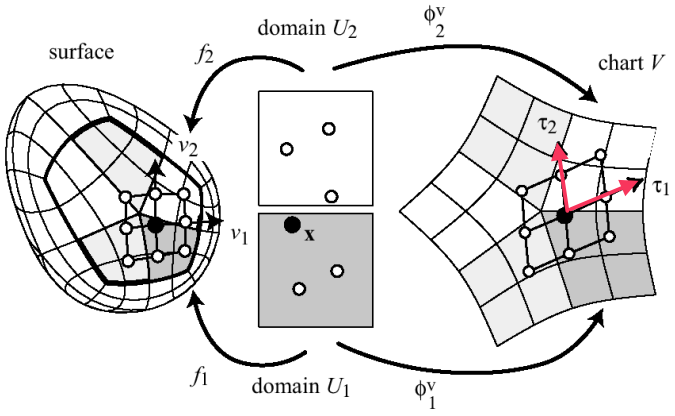
new york university mrl

Chart sampling



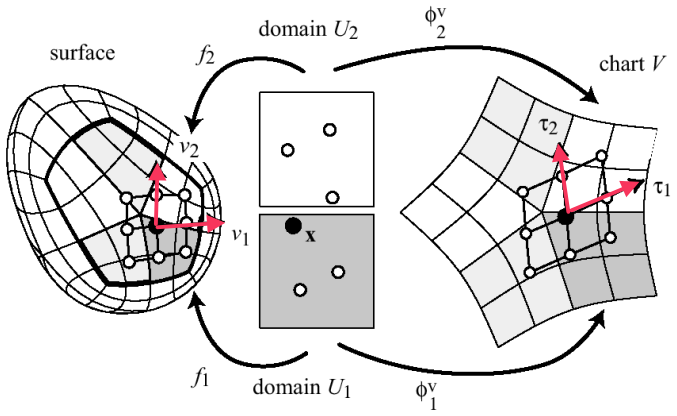
new york university mrl

Chart sampling



new york university mrl

Chart sampling



new york university mrl

Algorithm

Assume example texture has regular connectivity

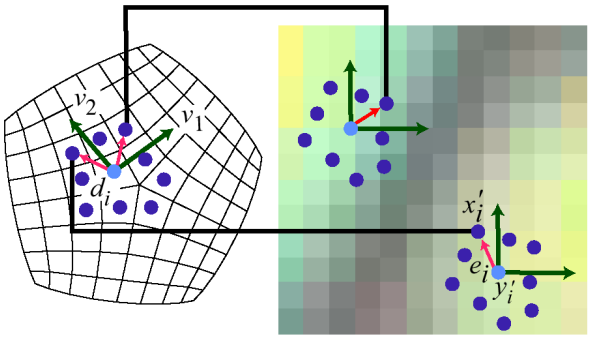
To synthesize sample x:

- pick a chart
- resample already synthesized texture on regular pattern using charts
- search for best match using TSVQ

new york university mrl

Inverse sampling

Sometimes need to resample example on irregular pattern



new york university mrl

Results

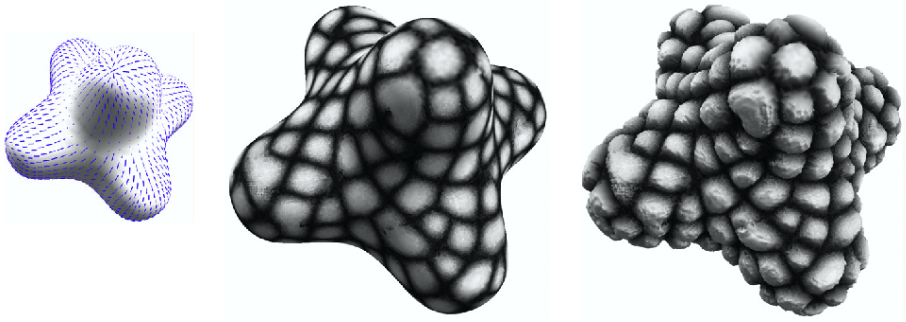
new york university mrl

Results

example multiscale coherent

new york university mrl

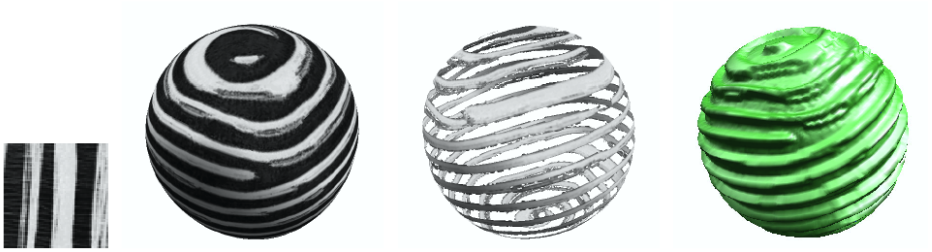
Results



synthesized geometry

new york university mrl

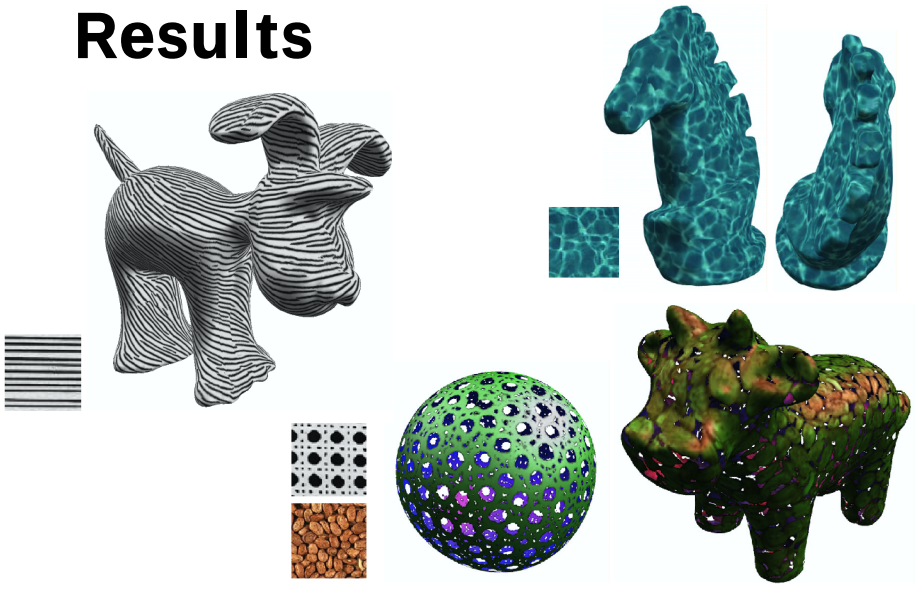
Results



transparency geometry

new york university mrl

Results



mrl

Summary

Adapting image processing algorithms to surface setting

- need to handle irregular sample patterns
- can use only local orientation
- need to take non-uniform density and anisotropy into account
- need to resample all the time
- no good sampling theory

new york university mrl

A Simple Algorithm for Surface Denoising

Jianbo Peng, Vasily Strela, Denis Zorin

New York University

Abstract

We present a simple denoising technique for geometric data represented as a semiregular mesh, based on locally adaptive Wiener filtering. The degree of denoising is controlled by a single parameter (an estimate of the relative noise level) and the time required for denoising is independent of the magnitude of the estimate. The performance of the algorihm is sufficiently fast to allow interactive local denoising.

1 Introduction

The complexity of the models used in computer graphics, visualization and geometric modeling applications constantly increases. It becomes more and more difficult to create such models by hand, and 3D scanning is emerging as an attractive alternative. However, the raw data produced by 3D scanners (range images or point clouds) are usually far from usable in any application. Considerable number of algorithms were developed for processing such data. A typical processing pipeline includes several stages:

- registration of raw data to create a single point cloud;
- conversion of the point cloud to an arbitrary fine polygonal mesh;
- decimation and reparameterization of the resulting mesh.

While the reparameterization step is not essential for every application, it is often desirable when a complex model has to be modified, stored and displayed interactively. As it was recently shown [10, 9], reparameterization combined with correctly chosen compression techniques results in substantial reduction in error (by factor of four) for compressed geometry compared to methods preserving fine mesh connectivity.

This result is not surprising if we consider the geometric data represented by a mesh from the informational point of view. In contrast to images, there are three distinct types of information associated with a mesh: connectivity (which vertices are connected by edges), geometry (vertex positions sampled from the original surface), and topology (topological structure of the surface represented implicitly by connectivity). It should be noted that only geometry and topology carry information about the original surface. Connectivity is not explicitly present in the original model and is introduced as an artifact of the algorithms used to convert the point cloud to a mesh.

If we adopt this point of view, there are three types of noise present in the mesh data:

- Connectivity noise, which is the pretty much all of the connectivity information for surfaces of low genus. As the only information about the original surface carried by connectivity is the topological information, connectivity can be replaced by any other as long as topology is preserved. All topological information we theoretically need for a surface of genus 0 can be represented by a tetrahedron.
- Topological noise, which is created by the algorithms used to extract a mesh from the point cloud.
- Geometric noise, due to the errors in measurement and resampling of the data at various processing stages.

Topology-preserving reparameterization can be thought of as removing connectivity noise; recent work [8] addresses the problem of topological noise. We focus on geometric noise removal, assuming that the surface is already reparameterized. While our method can be potentially applied before reparameterization, it works best and is most natural for semiregular meshes.

Reparameterization greatly simplifies the problem, because the surface can be considered as a function, and simple and efficient signal processing approaches can be applied. If reparameterization is ultimately performed on geometric data we believe that denoising is best left to the last stage, because additional noise can be introduced at the resampling stage. This is the case when our approach applies. If reparameterization will not be performed, more complex techniques for denoising on arbitrary meshes [2, 6] are more appropriate.

The algorithm that we propose is based on recent work in image denoising which uses locally adaptive Wiener filtering [16, 17, 21, 22]. The subbands of a multiscale representation are modeled as a product of a Gaussian random vector with a hidden multiplier variable. Estimation of the multiplier leads to the estimation of the local variance and allows standard Wiener denoising. The resulting algorithm is quite efficient, as it requires only a single pass over the surface at each resolution level. It is controlled by a single user-defined parameter, namely, an estimate of the noise magnitude. The performance does not depend on the magnitude of this estimate, i.e. strong noise reduction takes exactly as much time as moderate amount of denoising. Given an infrastructure for supporting semiregular meshes it takes very little time to implement (several hundred lines of code) and can be used for interactive local denoising of a model.

1.1 Denoising

We start with formulating the problem more precisely. *Given a surface corrupted by geometric noise, our goal is to produce a new surface which is as close as possible to the original one*. This task requires implicit or explicit assumptions (model) about the noise and the surface.

It is useful to consider a simple 1D example to understand the problem more clearly. If nothing is known about a 1D signal, it cannot be denoised. However, if the signal contains no frequencies above ω and the source of noise produces only frequencies above ω, low-pass filtering is an ideal denoising procedure. This is a simple example of general pattern common for a wide class (but not all) of denoising approaches: apply a transformation to represent the signal in a domain (in our case, frequency domain) where the noise is well separated from the signal, use assumptions on the structure of the transform coefficients of the signal and noise in order to remove the noise, apply the inverse transform.

For real-world signals, the situation is more complex: these signals typically have spectra overlapping the spectrum of the noise, and low-pass filtering is likely to remove important parts of the signal together with noise. Surface smoothing does precisely that for surfaces [23, 4]. The way to achieve better results is to use additional information about the properties of the signal. A classical example is wavelet thresholding methods for image processing [7]. These methods take advantage of the fact that wavelet bases have good compression properties: in such bases, a typical non-noisy

image will have mostly small coefficients, and only few large ones. Eliminating the small coefficients does not alter the reconstructed image much. One reason for this is that natural images often consist of large smooth areas (fine-level coefficients are small) separated by sharp boundaries (fine-level coefficients are large), with boundaries occupying only small area in images.

In contrast, the coefficient magnitude for noise is uniform and, as the signal energy is distributed over a large number of coefficients, each coefficient is likely to be small. This leads to the simple basic algorithm: apply a wavelet transform, threshold the coefficients and apply the inverse transform. For a restricted class of signals corrupted by white Gaussian noise a version of this procedure was proven to be optimal [7]. Note that this procedure is likely to preserve sharp transitions in a signal (edges for images, creases for surfaces).

Nevertheless, it is clear that a part of the useful signal is still removed. One can do better, however, by using additional assumptions. It was shown that Gaussian scale mixture (GSM) model is very suitable for than the statistics of wavelet coefficients of natural images [24, 22]. The combination of this model with Wiener filtering leads to better recovery of the original image. The resulting algorithm is not much more complex than the wavelet thresholding described above — the only additional step involved is local estimation of the signal variance. In the case of Gaussian noise the procedure is nearly optimal.

It turns out that GSM models also appear to reflect properly the statistics of multiscale representations of surfaces. Thus it is natural to apply the GSM-based denoising procedures to surfaces. The algorithm we propose is based on the general ideas of the image denoising algorithms but significantly differs from them in a number of aspects as detailed below.

2 Previous Work

Our algorithm primarily on work in image processing; relatively little has been done on surface denoising. Recent results include Clarenz et al. [2] on denoising of arbitrary meshes and Desbrun et al. [5] on denoising hight fields. In both cases, anisotropic curvature diffusion techniques are used. Our method is fundamentally different and difficult to compare directly to the diffusion-based approaches. Diffusion-based denoising is best regarded as a combination of smoothing and edge enhancement. It is relatively difficult to predict the scale of the noise that will be removed, and the amount of denoising depends on the algorithm running time. At the same time, as demonstrated in [2], curvature flow-based methods can be used on arbitrary meshes, while we assume reparameterization on semiregular meshes. For certain choices of parameters, our method produces results similar to anisotropic curvature diffusion. The methods are compared in greater detail in Section 7.

Recent developments in image denoising show that locally adaptive Wiener filtering is a very powerful technique. This approach was first developed in pixel domain [12, 11] and then extended to the multiresolution domain [16, 17] which allowed further improvement of the results. Local Wiener filtering uses a local estimate of the variance in either the spatial or the multiresolution domain. Wainwright and Simoncelli proposed a model that allows easy estimation of local variance and captures well the local statistical properties of wavelet coefficients of natural images [24]. This model is based on the class of random variables known as Gaussian scale mixtures (GSM). In the GSM model, groups of wavelet coefficients correspond to a product of a Gaussian random vector with a hidden multiplier variable. Similar models have been independently proposed in [15, 3]. The GSM approach combined with Wiener filtering was successfully implemented for image denoising [22]. We suggest a similar technique for noise removal on natural surfaces.

3 Overview of the algorithm

Our denoising procedure follows a common pattern described in Section 1.1. First, we apply a multiresolution transform described in Section 4 to a given noisy surface. We then use the GSM statistical model of the transform coefficients to distinguish the noise from the signal. The details of this step are given in Section 5. Finally, we reconstruct the surface from the denoised coefficients. See Section 6 for the complete description of the algorithm.

4 Multiresolution Surfaces

In this section, we describe in greater detail our assumptions about the parametric surface representation, and the specific representation we use.

It is generally sufficient to assume that the initial mesh was reparameterized on a mesh with semi-regular connectivity. The connectivity of such meshes can be obtained if we start with a relatively coarse mesh, and refine each face of such mesh regularly, in the simplest case, by recursive quadrisection of faces. The latter, however, not essential for our algorithm: any regular refinement can be used.

As a starting point, we use a Laplacian-pyramid multiresolution representation based on Loop subdivision . We refer the reader to [18, 25] for the details of implementation. The surface is represented by the coarsest level and the details at each level of resolution. The process of converting the finest-resolution data to the sequence of detail sets and the coarsest level mesh is called *analysis*. The process of reconstructing a surface from the coarse mesh and details is called *synthesis* The two processes are applied recursively, with analysis proceeding from finer to coarser levels, and synthesis from coarser to finer. A single step of both processes is illustrated in Figure 1.

For analysis, a smoothing filter is required in addition to subdivision rules. We use a simple Laplacian filter for smoothing.

It is important to note that the details at a finer level of resolution are represented in local frames computed from the previous coarser level. This is a valuable feature for surface editing and a natural way to represent surfaces: details are separated into tangential and normal components and become invariant witrh respect to rigid transforms. However, addition of the local frame makes the transform nonlinear. Our comparison of denoising with and without local frame transformations was inconclusive: it is still unclear if there is a substantial advantage in using local frames other than having a geometrically invariant result.

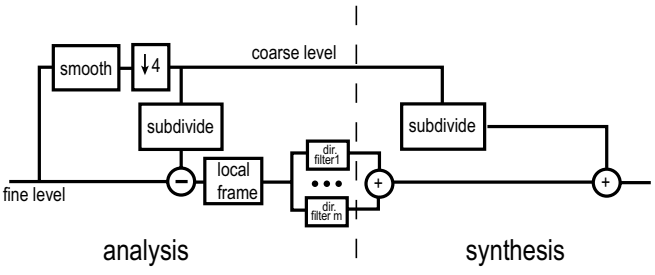


Figure 1: Synthesis and analysis diagrams for multiresolution surfaces.

We use an important modification to the pyramid based on the idea of steerable pyramids [20, 19]: a single detail band is decomposed into multiple directional bands, using directional filters (Figure 1). The number of directional bands m can be chosen arbitrarily by choosing the angular step θ_m. To reconstruct the signal the

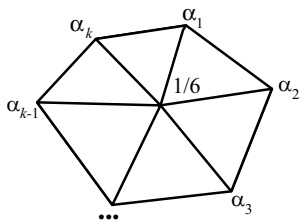


Figure 2: Directional decomposition of the details and the filter mask for a directional filter, $\alpha_i = \cos(\frac{2i\pi}{k} - \theta_m)$.

directional bands are simply added up to produce the detail. Introducing these directional bands is a crucial element of the algorithm. Clearly, the subbands are interdependent, and we need to store only two to be able to reconstruct the result of filtering in any direction.

5 GSM Model for Denoising Multiscale Data

As in the case of natural images, marginal distributions of the multiresolution coefficients of natural surfaces turn out to be sharp-peaked at zero and heavy-tailed (see Figure 3). The peak at zero is produced by the smooth regions, while heavy tails correspond to the slow decay of the coefficients at the edges. We propose to model this distribution by a Gaussian scale mixture process. The GSM random variables include several well-known sharp peaked and heavy tailed distributions such as generalized Gaussians, the α-stable family, and the Student t-variables [24]. One would expect a GSM model to be a good approximation in our case.

We now describe GSM in detail. A random vector X is said to be a Gaussian scale mixture if it is a product of two random variables: $X = \sqrt{z}U$, where z is a positive scalar random variable and U is a zero-mean Gaussian random vector with covariance C_u [1]. U and z are assumed to be independent. The probability density of a GSM variable is:

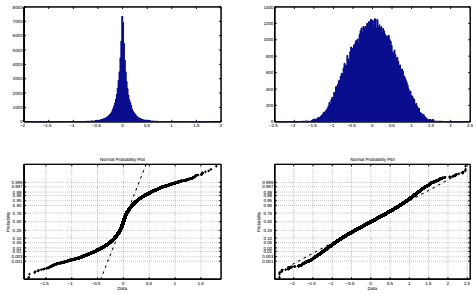
$$\mathcal{P}_x(X) = \int \frac{1}{(2\pi)^{N/2}\,|zC_u|^{1/2}} \exp\left(\frac{-X^T C_u^{-1} X}{2z}\right) \mathcal{P}_z(z)\,dz, \tag{1}$$

where N is the length of vectors U and X. Notice that normalized GSM variable $X/\sqrt{z}$ is Gaussian distributed which allows easy estimation of the statistical properties of the data. In particular, the Wiener filtering of the noisy GSM data should be close to optimal.

We assume that the directional detail coefficients in a single-ring neighborhood of a vertex on each level of a multiresolution mesh follows the GSM model. We also assume independence of the multipliers corresponding to different neighborhoods, even though the neighborhoods are overlapping. Moreover, in order to simplify the computations we treat both the coefficients of the noise and the signal in each neighborhood as uncorrelated (but not necessarily independent) and set their covariance matrices to be multiples of the identity $C_u = \sigma_u^2 I$, $C_w = \sigma_w^2 I$; we assume the variance of noise known (in practice, it is estimated by the user). While it is possible to vary σ_w, we use a single value for the whole surface which is a reasonable assumption for scanned models.

One can test how well the GSM model describes actual data. Let X be a vector corresponding to a single ring of pyramid coefficients around a vertex of a "clean" surface, with x the coefficient at the center of the ring. If the model is correct and there is a good estimate $\widehat{z}(X)$ of $z(X)$ then the distribution of the normalized coefficient $x' = x/\sqrt{\widehat{z}(X)}$ should be close to a Gaussian. We use the maximum likelihood estimator for the multiplier [24, 16, 22]:

$$\widehat{z}(X) = (X^T C_u^{-1} X)/N.$$



before dividing by $\sqrt{\widehat{z}}$ after dividing by $\sqrt{\widehat{z}}$

Figure 3: Histograms and normal probability plots of the pyramid coefficients before and after normalization of each coefficient by the estimate of hidden multiplier $\sqrt{\widehat{z}}$.

Within our assumptions $C_u = \sigma_u^2 I$ and $\sigma_u^2 \widehat{z}(X) = \frac{X^T X}{N}$ is just a local estimate of the variance of x. Figure 3 shows the marginal histograms and normal probability plots of x and x'. Presented data comes from one component of the third level of the multiresolution decomposition of the denoised surface (the model of a dog); similar results were obtained for a number of other scanned models. The histogram of the normalized coefficients is nearly Gaussian, and its corresponding normal probability plot lies nearly along a line. Thus the GSM model does a reasonable job approximating the data.

Our main goal is to estimate the multiplier z in the presence of noise. This will allow us to compute the variance of each element and use the Wiener filter to remove the noise. Suppose that vector Y is obtained from X by adding Gaussian white noise with variance σ_w^2 and mean 0, $Y = X + \sigma_w W$. If X is a GSM vector then each observed noisy coefficient can be represented as $y = \sqrt{z}u + \sigma_w w$, where σ_w^2 is the variance of the noise and w has Gaussian distribution with variance 1 and mean 0. If the value of z were known, then y would also be Gaussian distributed, and the optimal estimate of x would be the linear (Wiener) solution:

$$\widehat{x} = \frac{z\sigma_u^2}{z\sigma_u^2 + \sigma_w^2}\, y. \tag{2}$$

We use the maximum likelihood estimator in order to obtain $\widehat{z}(Y)$, $\widehat{z}(Y) = \left(1/\sigma_u^2\right)\left(Y^T Y/N - \sigma_w^2\right)$. The derivation of this result is given in [16]. When applying this formula to the real data, one often gets a small negative value for $\widehat{z}(Y)$. This happens because the neighborhood is not large enough to capture the statistics of the data or the estimated noise level is too large. In this cases we set $\widehat{z}(Y)$ to zero. We estimate the variance of the center of a neighborhood Y as

$$\widehat{z}\sigma_u^2 = \max\left(Y^T Y/N - \sigma_w^2, 0\right). \tag{3}$$

Equations (2), (3) are used in our denoising algorithm.

6 Denoising Algorithm

We implemented the results of previous sections in the denoising procedure. It consists of there steps: 1) multiresolution decomposition (Section 4). 2) Noise removal using formulas (2), (3) on each level of the decomposition. 3) Reconstruction.

To use formulas (2) and (3) one needs to know the variance of the noise σ_w^2. This is the parameter supplied by the user. In Section 7, we show the results for various values of σ_w. It is also possible to choose different numbers m of directional components for the filters, but, not surprisingly, 6 is the best choice for semiregular triangular meshes.

Denoising Algorithm

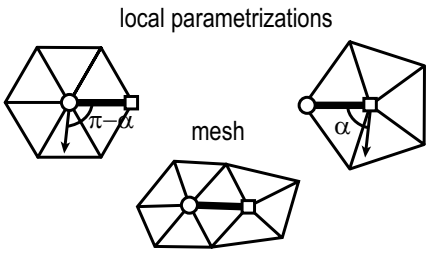


Figure 4: Local alignment of directions

1. Perform analysis of the mesh as described in Section 4.
2. For the details at each level of the decomposition
a) Compute the aligned directional components of the neighborhood of each vertex in the detail mesh in m directions.
b) Estimate the variance of each directional coefficient using formula (3); if the result is negative set the variance to zero.
c) Replace each directional coefficient by its Wiener estimate (2).
d) Replace the value of the center with the combination of the m denoised directional coefficients.
3. Reconstruct the surface mesh.

Unlike images, for which directional components can be chosen to be oriented consistently along global directions, for general surfaces this is not possible. However, as our algorithm is local, only local alignment is required. To choose the aligned directional components, we assume that the single-ring neighborhood of a vertex at a fixed refinement level is parameterized over a k-gon (Figure 4). As shown in the figure, for an arbitrary edge fixed as the zero direction one can pick corresponding directions for the filters for surrounding vertices. The result of filtering in one of these directions is simply a linear combination of two directional details.

7 Results and Discussion

Results of applying our algorithm to data with a high level of artificial noise added for several settings of σ_w are shown in Figure 5. Other denoising results are shown in Figures 9–10. The input parameter σ_w (estimate of the noise level) was chosen as a percentage of the average distance between the initial surface and the coarsest surface. Timings are provided for a relatively slow machine (200 MHz SGI Indigo2).

Comparison with anisotropic curvature diffusion. For comparison, we have implemented anisotropic curvature diffusion as described in [2]. Figure 9 demonstrates that for a certain choice of estimated noise value our algorithm produces results visually similar to anisotropic curvature diffusion [2, 5, 6]. The image was chosen to be as similar as possible to the one shown in [2]. The approaches based on statistical models (such as ours) and deterministic approaches (such as anisotropic curvature diffusion) are based on very different principles and, from mathematical point of view, solve different problems; hence it is difficult to compare the algorithms quantitatively.

The difference between the algorithms merits detailed discussion. The idea of anisotropic curvature diffusion can be summarized as follows: the denoised surface is obtained as the solution at some time τ of an anisotropic diffusion equation. The diffusion tensor is anisotropic near edges, with zero diffusion perpendicular to the edge and maximal along the edge. The edges are detected at each time step using a principal curvature threshold, applied to the curvature values obtained for a smoothed version of the surface. There are three parameters determining the result: the time τ, the constant ϵ controlling the amount of smoothing used before curvatures are computed and λ, the edge detection threshold; ϵ has less impact on the result, so we restrict our attention to τ and λ.

Feature preservation. Both algorithms attempt to preserve important surface features. Anisotropic curvature diffusion detects and attempts to preserve and sharpen edges [2, 6]. Our algorithm has implicit edge detection build in: if there is an edge passing through a point in a certain direction, in orthogonal direction the variance will be considerable and Wiener filtering will not reduce these coefficients by much if at all. The advantage of our approach is that there is no global threshold λ for curvature-controlling edge detection; this parameter is difficult to pick. This can be also regarded as disadvantage as there is no direct edge detection control. The best our algorithm can do is to preserve the noise perpendicular to the edge near the edge; anisotropic curvature diffusion can enhance edges. This is useful in the cases of man-made objects for which a collection of smooth surfaces with sharp edges is a good model. This is less useful and can be harmful for natural object, which seldom have sharp edges. For such objects increasingly sharp edges tend to appear at random locations.

Generality. Our algorithm relies on the multiresolution structure of the mesh, hence applies only to models that were reparameterized on semiregular meshes. In contrast, curvature diffusion works on an arbitrary mesh. while it might be possible to generalize our algorithm to hierarchies on irregular meshes, this would make it significantly more complex.

Running time. With anisotropic curvature diffusion, if a large amount of denoising is desired, large values of τ should be used and the algorithm takes longer, even if implicit integration and large time steps are used. Our algorithm takes exactly the same time no matter how much denoising is desired. For the specific example shown in Figure 9, our conjugate-gradient implementation of curvature diffusion is significantly slower than the GSM algorithm: (260 sec. vs. 16 sec.) However, an efficient multigrid solver is likely to make the times much closer. Our algorithm is fast enough to enable interactive applications.

Locality. Our algorithm can be easily applied locally (the video submitted with the paper shows an interactive application when the noise is removed locally). It is not clear how anisotropic diffusion will behave when applied locally.

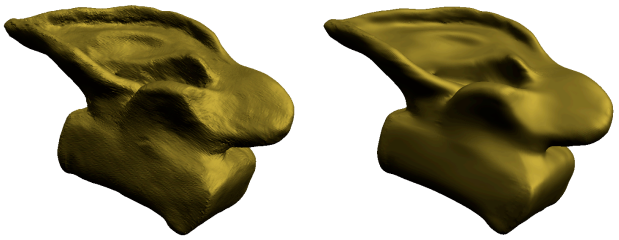
Ease of implementation. As we have mentioned, our algorithm takes very little effort to implement: it is a simple iteration over vertices with filters applied to the immediate neighbors. By comparison, curvature diffusion requires a good solver running on arbitrary meshes to be efficient, and even the basic algorithm requires more work.

8 Conclusions

We have presented a new algorithm for denoising of natural surfaces. It is based on a multiresolution steerable decomposition and utilizes a GSM statistical model of the transform coefficients. The results of our experiments are quite encouraging and compare favorably with other techniques; our algorithm ensures noise removal while preserving essential geometrical features.

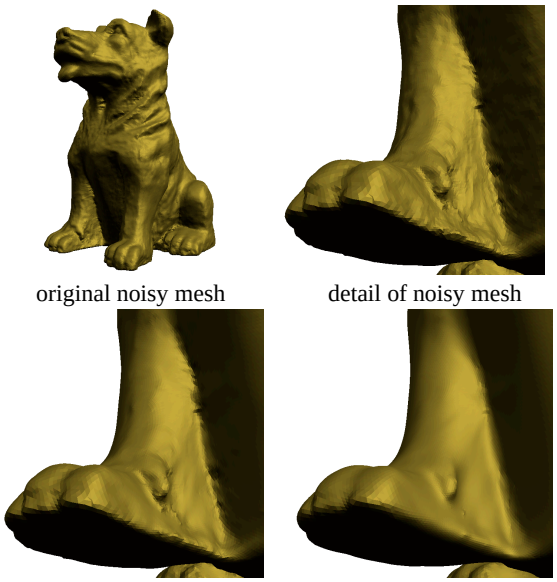
Our results are just a first step in applying the GSM model for the denoising of surfaces. One can employ different multiresolution decompositions and extend the denoising algorithm along the lines suggested in [22]. In particular a non-trivial covariance structure of the coefficients within each neighborhood can be used.

Our algorithm is based on the GSM assumption on the statistics of the multiscale coefficients. We have observed that it is a reasonable assumption for several models, but clearly more extensive studies are necessary.



original noisy mesh $\sigma_w = 2.8\%$

Figure 9: Denoising a scanned and parameterized model of an ear (335 thousand triangles, 360 thousand after parameterization).



original noisy mesh detail of noisy mesh

$\sigma_w = 2.5\%$ $\sigma_w = 7.8\%$

Figure 10: Denoising a scanned and parameterized model of a dog for different choices of σ_w (264 thousand triangles, 391 thousand triangles after parameterization). Denoising time: 59 secs.

Many questions which have well-understood answers for images (e.g. measure of difference between images) are much more difficult for surfaces and require further study to make it possible to compare algorithms in a more quantitative manner.

Note on Figure 7: We believe the small scale random texture visible on the surface is an artefact of the scanning and reconstruction process; it is similar in scale to the texture we have observed on other objects scanned using Cyberware scanners; according to Levoy et al. [13] the scale of the details on the surface of polished marble is less than 30 nm, which is far less than the characteristic texture size. On the other hand, the texture is too random and too closely spaced (0.5-1 mm is the characteristic scale) to be chisel marks which are more likely to be at least 2 mm wide [14].

References

[1] D Andrews and C Mallows. Scale mixtures of normal distributions. *J. Royal Stat. Soc.*, 36:99–, 1974.

[2] U. Clarenz, U. Diewald, and M. Rumpf. Anisotropic geometric diffusion in surface processing. In *Proceedings of Visualization 2000*, 2000.

[3] M S Crouse, R D Nowak, and R G Baraniuk. Wavelet-based statistical signal processing using hidden Markov models. *IEEE Trans. Signal Proc.*, 46:886–902, April 1998.

[4] M Desbrun, M Meyer, P Schroder, and A Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Siggraph'99*, 1999.

[5] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Anisotropic feature-preserving denoising of height fields and bivariate data. In *Proceedings of Graphics Interface 2000*, 2000.

[6] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Discrete differential-geometry operators in nd. submitted, 2000.

[7] D Donoho. Denoising by soft-thresholding. *IEEE Trans. Info. Theory*, 43:613–627, 1995.

[8] Igor Guskov and Zoë Wood. Topological noise removal. In *Proceedings of Graphics Interface 2001*, 2001.

[9] Andrei Khodakovsky and Igor Guskov. Normal mesh compression. submitted for publication.

[10] Andrei Khodakovsky, Peter Schröder, and Wim Sweldens. Progressive geometry compression. *Proceedings of SIGGRAPH 2000*, pages 271–278, July 2000. ISBN 1-58113-208-5.

[11] D T Kuan, A A Sawchuk, T C Strand, and P Chavel. Adaptive noise smoothing filter for images with signal-dependent noise. *IEEE Pat. Anal. Mach. Intell.*, PAMI-7:165–177, March 1985.

[12] J S Lee. Digital image enhancement and noise filtering by use of local statistics. *IEEE Pat. Anal. Mach. Intell.*, PAMI-2:165–168, March 1980.

[13] M. Levoy. Digital michalangelo project web site. `http://www.graphics.stanford.edu/projects/mich/other-body-parts/other-body-parts.html`.

[14] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. *Proceedings of SIGGRAPH 2000*, pages 131–144, July 2000. ISBN 1-58113-208-5.

[15] S M LoPresto, K Ramchandran, and M T Orchard. Wavelet image coding based on a new generalized gaussian mixture model. In *Data Compression Conf*, Snowbird, Utah, March 1997.

[16] M K Mihçak, I Kozintsev, K Ramchandran, and P Moulin. Low–complexity image denoising based on statistical modeling of wavelet coefficients. *IEEE Trans. on Signal Processing*, 6(12):300–303, December 1999.

[17] P Moulin and J Liu. Analysis of multiresolution image denoising schemes using a generalized Gaussian and complexity priors. *IEEE Trans. Info. Theory*, 45:909–919, 1999.

[18] K. Pulli and M. Lounsbery. Hierarchical editing and rendering of subdivision surfaces. Technical Report UW-CSE-97-04-07, Dept. of CS&E, University of Washington, Seattle, WA, 1997.

[19] E P Simoncelli and W T Freeman. The steerable pyramid: A flexible architecture for multi-scale derivative computation. In *Second Int'l Conf on Image Proc*, volume III, pages 444–447, Washington, DC, October 1995. IEEE Sig Proc Society.

[20] E P Simoncelli, W T Freeman, E H Adelson, and D J Heeger. Shiftable multiscale transforms. *IEEE Trans Information Theory*, 38(2):587–607, March 1992. Special Issue on Wavelets.

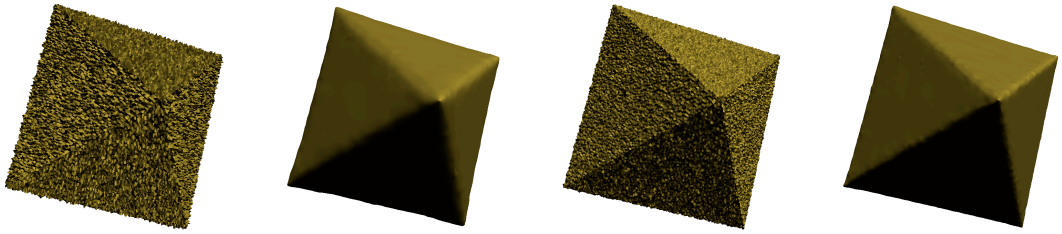
[21] V Strela. Denoising via block Wiener filtering in wavelet domain. In *3rd European Congress of Mathematics*, Barcelona, July 2000. Birkhäuser Verlag.

[22] V Strela, J Portilla, and E Simoncelli. Image denoising using a local gaussian scale mixture model in the wavelet domain. In *Proc SPIE, 45th Annual Meeting*, San Diego, July 2000.

[23] G Taubin. A signal processing approach to fair surface design. In *Siggraph'95*, pages 351–358, 1995.

[24] M J Wainwright and E P Simoncelli. Scale mixtures of Gaussians and the statistics of natural images. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Adv. Neural Information Processing Systems*, volume 12, pages 855–861, Cambridge, MA, May 2000. MIT Press.

[25] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. *Proceedings of SIGGRAPH 97*, pages 259–268, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.



noisy mesh $\sigma_w = 12\%$ noisy mesh $\sigma_w = 4.3\%$

Figure 5: Denoising a simple mesh with artificial noise for different choices of estimated noise σ_w (98 thousand triangles).

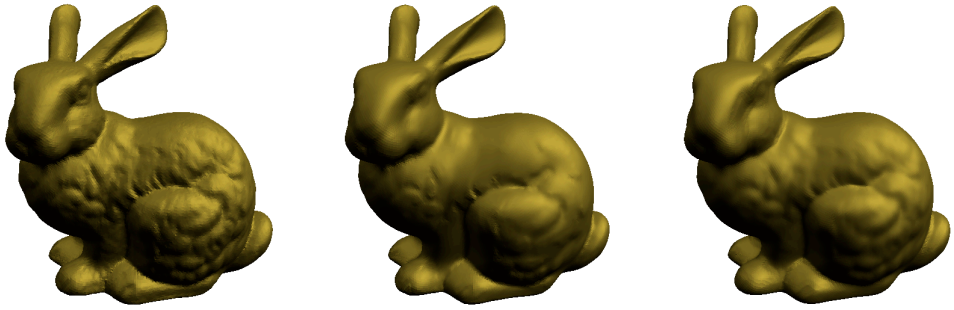
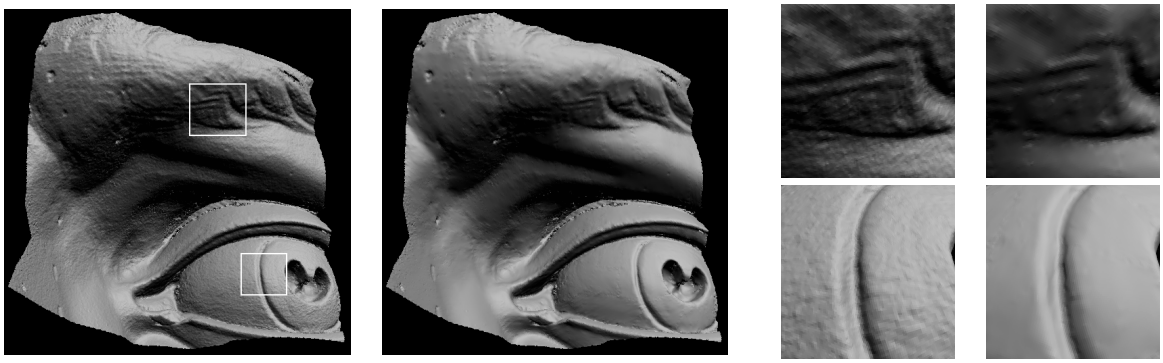


Figure 6: Denoising the reparameterized Stanford bunny mesh. (71 thousand triangles, 145 thousand after parameterization). From left to right: the reparameterized mesh; denoised by our algorithm; by anisotropic geometric diffusion method.



noisy mesh $\sigma_w = 1.4\%$ details

Figure 7: Denoising a part of the model of Michalangelo's David. (scanned at 0.29mm resolution; unstructured mesh 0.63 mln. triangles, 1,2 mln. triangles after parameterization). Denoising time: 167 secs. Original mesh courtesy of Marc Levoy, Stanford Computer Graphics Lab. From left to right: original mesh; denoised mesh; magnified views of two areas on the mesh before and after denoising. Note that chisle marks in the first area are preserved, while the small-scale noise is removed.

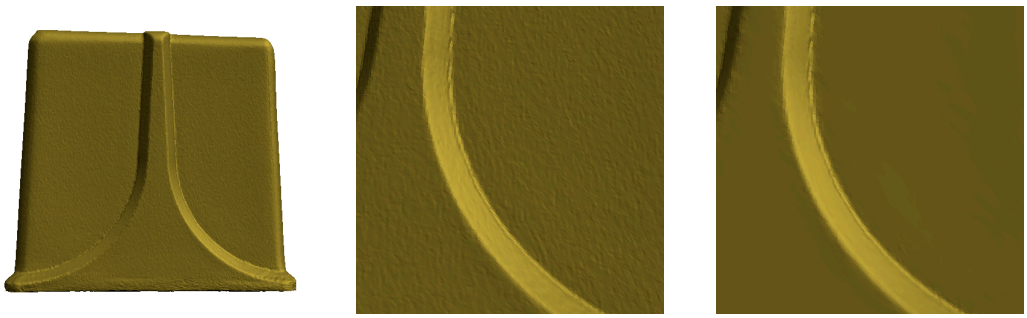


Figure 8: Denoising a scanned and parameterized model of a Ascension Technologies transmitter. From left to right: original model; magnified view of an area before and after denoising. Note that in the flat areas all small scale features were removed, with almost no change at the creases.

Texture and Shape Synthesis on Surfaces

Lexing Ying, Aaron Hertzmann, Henning Biermann, Denis Zorin
New York University
http://www.mrl.nyu.edu

Abstract. We present a novel method for texture synthesis on surfaces from examples. We consider a very general type of textures, including color, transparency and displacements. Our method synthesizes the texture *directly* on the surface, rather than synthesizing a texture image and then mapping it to the surface. This approach avoids many problems associated with texture mapping, such as seams, distortion, and repeating patterns. The synthesized textures have the same qualitative visual appearance as the example texture, and cover the surfaces seamlessly, without distortion. We describe two synthesis methods, based on the work of Wei and Levoy and Ashikhmin; our techniques produce similar results but directly on surfaces.

1 Introduction

Computer graphics applications increasingly require surfaces with highly detailed reflective properties, geometry and transparency. Constructing such detailed appearances manually is a difficult and tedious task. A number of techniques have been proposed to address this problem; procedural synthesis techniques are among the most widely-used. Most recently, a number of techniques [5, 18, 3] make it possible to synthesize textures for objects from examples.

Creating a surface with a complex appearance can be viewed as synthesis of a collection of functions on an arbitrary two-dimensional domain. (This assumes that the topology of the surface does not change, but the geometry can change). These functions typically include color, transparency, normals and coordinates of surface points. We will refer to all such functions as textures. Note that textures can be thought of as continuous and defined directly on surfaces, although they will be represented as samples in implementation. In this view, previous example-based texture synthesis algorithms synthesize attributes for a special kind of surfaces: flat rectangles.

In this paper, we show how synthesis from examples can be extended to synthesis on surfaces. The obvious approach to the problem, synthesizing an attribute on a rectangle and mapping it to an arbitrary surface, produces many artifacts (e.g. seams) and requires solving a fundamentally difficult problem of mapping a rectangle to an arbitrary surface with minimal distortion. Typically, creating such maps requires considerable user intervention. In general, performing synthesis directly on a surface will give better results than synthesizing a texture on a rectangle and then mapping it to a surface.

Existing texture synthesis methods rely on the presence of identical, regular sampling patterns for both the example and the synthesized texture. Therefore, it is impossible to apply such methods directly to surfaces. In this paper, we regard the example and the synthesized texture as continuous functions that happen to be represented by samples, but not necessarily laid out in identical patterns. Whenever necessary, we resample either the example or the synthesized texture on a different pattern.

We describe two specific synthesis methods, based on the methods of Wei and Levoy [18] and Ashikhmin [3]. As with image synthesis, the choice of algorithm depends primarily on the texture.

The main contributions of this paper are: Generalizations of existing image texture synthesis methods to synthesis on surfaces; synthesis of surface texture maps independent of parameterization; efficient and accurate neighborhood sampling operations; and synthesis of texture, transparency, and displacements.

2 Previous Work

The problem of synthesizing texture from examples has recently enjoyed a renaissance of research interest. A popular approach from the vision community is Markov Random Fields (MRF) modeling, in which one models the joint probability densities of image features (e.g. oriented Gabor filter responses or steerable filter responses) from a texture (e.g. Zhu et al. [19], Portilla and Simoncelli [14]). However, MRF algorithms typically require a time-consuming optimization in order to create an image with statistics that match the model. Heeger and Bergen [9] model texture in terms of filter histograms, and produce textures with matching histograms.

Recently, several nearest-neighbor methods have been shown to produce high quality textures; these algorithms may be viewed as approximating an MRF. De Bonet [4] demonstrates a coarse-to-fine procedure that fills in pixels in an output texture image pyramid by copying pixels from the example texture with similar coarse image structure. Efros and Leung [5] create textures in a single scale using a single-scale neighborhood. Wei and Levoy [18] combine these methods, by using a neighborhood that contains both coarse-scale and same-scale pixel information. Tree-structured vector quantization (TSVQ) [6] is used to accelerate the search for the nearest neighbor. Ashikhmin [3] produces high-quality textures by copying contiguous texture patches when possible. This method is very fast, but may produce sharp image discontinuities in some situations.

Neyret and Cani [13] texture a surface isotropically by tiling from a small collection of tileable example texture triangles. Praun et al. [15] extend this by placing oriented texture patches over an independent parameterization of a surface. Although these methods produce high-quality results for many textures, they have some drawbacks: they cannot capture low-frequency texture without sacrificing high-frequency randomness, and the texture patches do not necessarily line up exactly, which requires blending to prevent discontinuities between patches.

Existing methods for example-based surface synthesis interpolate existing shapes via 3D morphing (e.g. [17, 1, 16]). In contrast, our work generates shapes that have similar statistics to examples.

3 Overview

Given a 2D example and a 3D surface, our goal is to create a texture that "looks like" it came from the same process that generated the example. We make the assumption that this is equivalent to requiring the texture on every small surface neighborhood to "look like" the texture on some neighborhood in the example.[1] The example and synthesized

[1]This formulation is based on the *Markov condition* assumption on the texture: we assume that the texture has local statistics (the value at a point depends only on its local neighborhood), and stationary statistics (the dependence is the same for every point). These assumptions imply that the the surface texture will look like

texture will be discretized into samples, although not necessarily with the same sample density. The texture may be from any domain: in particular, we explore image textures, transparency textures and geometric textures. For brevity, we refer to the synthesized texture as *target*.

Texture representation: For simplicity, we assume that the example texture is resampled on a rectangular sampling grid, i.e. that it is an image. The target texture is represented by samples on the surface. We assume that there is a collection of rectangular images mapped to the surface and the texture samples will be stored in these images.

An important feature of our approach is that the synthesis process is independent of the choice of the texture-mapping parameterization: given a parameterization, our method will synthesize a texture without distortion on the surface.

Images mapped to surfaces are usually called texture maps. However, using this term in our context would result in considerable confusion; we use the term *image maps* instead.

Idea of the algorithms: The basic idea of our algorithms is quite simple. For each sample of the target texture, we consider the previously-synthesized texture within a small neighborhood of the sample. Then we locate a neighborhood in the example on which the texture is similar and copy the value of the texture in the center of the neighborhood to the target sample.

Several issues need to be addressed to make this idea practical:

- how to pick samples representing a texture on a target neighborhood,
- how to compare neighborhoods in the target and example,
- how to find similar neighborhoods in the example.

The two methods that we describe use somewhat different approaches to these problems. Each method has its strengths and weaknesses, illustrated in Section 6.

Comparing neighborhoods: Both methods resample on a common sample pattern in order to make it possible to compare neighborhoods on the example and target.

As most interesting textures are anisotropic, orientation must be established on the surface. We use a vector field to specify the correspondence between orientation on the surface and orientation in the domain of the example texture. A pair of orthogonal tangent vector fields v_1 and v_2 is used for this purpose. To compare the texture on the neighborhood on the surface centered at x to a neighborhood centered at y in the example we establish a map between the neighborhoods, mapping x to y and v_1 and v_2 to the coordinate directions in the example.

The field v_1 is computed using the method described in [10][2]. The field v_2 is computed as the cross-product of v_1 and the oriented surface normal. The field could also be specified interactively, as in [15].

Synthesis methods: Our first method is based on [18] and described in Section 4. For each generated sample x, we attempt to find the neighborhood in the entire example that matches the neighborhood of x as closely as possible. The sampling pattern of the

the example texture if this joint density of texture values is the same for the surface as for the example.

[2]Since we are optimizing a vector field, and do not desire 90^o-invariance as in [10], the optimization formula is of the form $\sum \cos((\theta_i - \varphi_{ij}) - (\theta_j - \varphi_{ji}))$ instead of $\sum \cos 4((\theta_i - \varphi_{ij}) - (\theta_j - \varphi_{ji}))$.

example is used to resample the neighborhood of the target. The fixed sampling pattern makes it possible to accelerate the search with standard nearest-neighbors algorithms, such as TSVQ.

The second method, based on [3] (Section 5), selects only from example neighborhoods that are spatially coherent in the example with some already-synthesized sample near x (*coherent synthesis*). This makes it necessary to track the source of each generated sample. In this case, we found it possible to use the target sampling pattern for comparison of neighborhoods, which is simpler than resampling the target. This makes it impossible to use the acceleration techniques such as TSVQ, as the sampling patterns on the target vary greatly, but, since only a few neighborhoods will be tested, acceleration is unnecessary.

4 Multiscale Synthesis

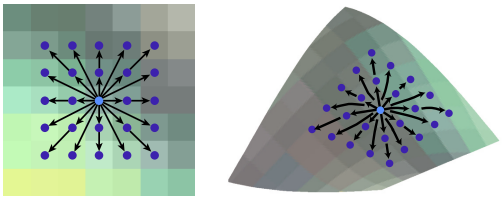
Our first method is based on the multiscale image synthesis procedure of Wei and Levoy [18]. In this method, we first synthesize a coarse version of the surface texture, and then perform coarse-to-fine refinement. This method allows us to efficiently capture both coarse and fine-scale statistics, and to perform several iterative refinements on the texture.

Our algorithm requires that the surface is covered by an atlas of sufficiently large overlapping charts, as explained below. We use the multiresolution subdivision surface representation and take advantage of the readily-available charts for such surfaces. There are several methods available for converting an arbitrary mesh to this representation [11, 8, 20]. However, such conversion is not fundamentally required for the algorithm: any method for constructing an atlas of charts can be used.

Our algorithm begins by creating a multiscale hierarchy (a Gaussian pyramid) for the example image and for each of the image maps in the target. Every level of the hierarchy will be synthesized, from coarse-to-fine.

For the coarsest level, we iterate over all image maps and iterate over coarse-level samples in each map (typically, very few samples per map). In order to synthesize a sample x, we sample the previously synthesized target texture in a neighborhood of x using the regular pattern of the example. Then we search the example to find the nearest match under a weighted l^2-norm. We weight samples with a Gaussian kernel in image space. The techniques that we use for sampling are described below. We then copy the central sample from the best-matching neighborhood to x. This brute-force search for the best matching neighborhood is rather inefficient; however, there are very few samples to synthesize or to search at the coarsest level of the pyramid, and so the total synthesis time at the coarse level is quite small. The very first sample that is synthesized is copied from a random location in the example texture.

We synthesize each of the remaining levels using a two-pass algorithm based on Wei and Levoy's [18] hole-filling algorithm. In the first pass, we synthesize each sample of a level of the hierarchy using a neighborhood that contains only samples from the coarser level of the pyramid. In the second pass, we refine the texture at the current level using the composition of the square neighborhood from the current level and one from the coarser level. This means that all samples in each neighborhood have already been synthesized, allowing us to use TSVQ [6] or Approximate Nearest Neighbors [2] to accelerate the nearest-neighbors searches in both passes. The best match found during these searches is copied to the target sample. We also introduce some randomness into the search, with the same randomization used in [4, 18]: We locate the eight nearest-neighbors matches found during the TSVQ traversal with backtracking,



(a) (b)

Fig. 1. Surface marching for neighborhood sampling. (a) A 5x5 rectangular surface neighborhood pattern on an image. (b) A corresponding sampling pattern on a surface. From the center sample, we "march" over the surface, in directions corresponding to the directions in the image plane. This gives us a set of surface sample locations. Values at each location are determined by bilinear sampling from the texture map. The orientation of the pattern is determined by the surface vector field.

discard all matches that have a distance worse than the best match by some factor, and then randomly pick one of the remaining matches by uniform random sampling.

We use two different methods for sampling neighborhoods on a surface. For the coarsest levels of the image hierarchy, we use *surface marching*, in which we traverse over the smoothed geometry to locate sample points. For the remainder of the image hierarchy, we perform *chart sampling*, in which we construct sampling neighborhood patterns in a globally-defined parametric domain. In our experiments, chart sampling gives better performs twice as fast; marching is only used for coarse levels where chart sampling cannot be used (for reasons described below).

4.1 Surface Marching

In the surface marching algorithm, we collect a grid of sample locations that corresponds to a grid of locations in the plane (Figure 1), using a tesselated mesh representation of the surface. For each sample point, we compute the angle and distance in the plane to the point. We then draw a straight path from the surface sample point in the computed distance and direction (with respect to the orientation field on the mesh) to find the corresponding surface sample point. When the path intersects a mesh edge, the line is continued on the adjacent face, at the same angle to the mesh edge as on the previous face.

There are several problems with this approach. First, it is not guaranteed to give even sampling of a surface neighborhood in irregular geometry. Second, the sampling pattern is numerically unstable, as minute surface variations can cause substantial variations in the pattern. Finally, this method is relatively slow, because of the many geometric intersection and projection operations required.

4.2 Chart Sampling

Rather than trying to move on the surface from one face to the next, one can take advantage of a suitable surface parameterization. Recall that our goal is to be able to sample the texture at (approximately) regularly spaced locations around a sample x. Suppose a sufficiently large part of the surface around x is parameterized over a planar domain V. Then we can sample in the parametric domain V, choosing the sampling pattern in such a way that the images of the sample points on the surface form the desired approximately regular arrangement (Figure 2). This can be achieved by using the Jacobian of the map g from V to the surface to predistort the sampling pattern.

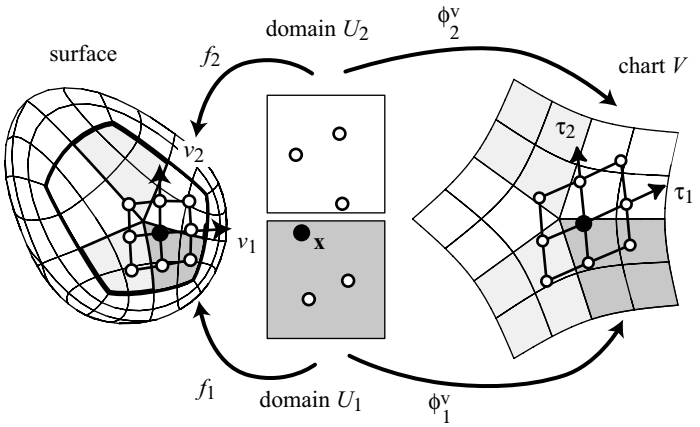


Fig. 2. Chart sampling. In order to sample the texture in the neighborhood of x, we construct its sampling pattern in the chart. The pattern is distorted in the chart such that it will be aligned with the surface orientation field (v_1 and v_2) and roughly square.

The crucial assumption of the method is that the size of the neighborhood to be sampled is sufficiently small, so that the parameterization for the neighborhood is sufficiently close to linear and each neighborhood fits on the image of one *chart* $g(V)$. When this does not hold, as happens at the coarsest levels of synthesis, chart sampling cannot be used and we perform marching instead.

We would like every point of the surface to be far from the boundaries of a chart, so we design the charts to overlap. There are many different ways to construct atlases of overlapping charts (e.g. [7]). We have chosen a way that works well for *multiresolution subdivision surfaces* [20]. It is important to note that we only use the associated chart construction in our algorithm. Charts can be also constructed for an arbitrary mesh using standard texture mapping techniques.

Chart sampling in detail: To explain more precisely how chart sampling is performed we need some additional notation. Suppose U_i are rectangular domains on which the surface is parameterized and f_i are the parameterizations (Figure 2). For a subdivision surface, we have one domain per face of the control mesh. In practice, f_i are represented by piecewise-linear approximations defined by arrays of vertices, but it is convenient to consider f_i and other maps as continuous for now. For the multiresolution subdivision surfaces, the *characteristic map* can be used to produce charts. Each chart corresponds to a vertex v. The part of the surface parameterized by the chart is defined on the union $\cup_i g(U_i)$ where i indexes all faces sharing v. The chart map g is defined implicitly by specifying a map Φ^v_i for each vertex, mapping U_i into a planar domain V. Then g is taken to be $g = f_i \circ (\Phi^v_i)^{-1}$ on $\Phi^v_i(U_i) \subset V$.

The values of the maps Φ^v_i are the values of the characteristic map; these values can be precomputed depend only on the valence of v. See [20] for further details on characteristic maps and how they can be computed.

Finally, we assume orthogonal unit tangent vector fields v_1 and v_2 are defined on all U_i; these fields may be discontinuous. These fields specify local orientation for synthesized unisotropic textures.

We use the notation Dh to denote the differential of a map $h : R^m \to R^n$, which is a linear map given by the matrix $(\partial h^i / \partial x^j)_{ij}$.

Now we are ready to define the procedure to compute samples. Given a sample x in the domain U_1, we wish to compute a set of samples x_{lm} in domains U_i such that their images $f_i(x_{lm})$ form a pattern close to a square grid with step δ in each direction, centered at $f_1(x)$. The samples are computed in several steps (Figure 2):

1. Map x to the chart domain V: $y = \Phi_1^v(x)$.
2. Compute vectors τ_1 and τ_2, such that the differential $Df_1(D\Phi_1^v)^{-1}$ maps τ_1 to v_1 and τ_2 to v_2; As v_1 and v_2 are tangent vectors, they can be written as $Df_1 w_j$ $j = 1, 2$, for some two-dimensional w_j. Then, more explicitly, $\tau_j = D\Phi_1^v w_j$, $j = 1, 2$.
3. Compute samples in the chart domain: $y_{lm} = y + l\tau_1\delta + m\tau_2\delta$, for $l, m = -2..2$ (a 5x5 sampling pattern).
4. Map them back to one of the parametric domains U_i, depending on which part $\Phi_i^v(U_i)$ of the chart domain V y_{lm} is located: $x_{lm} = (\Phi_i^v)^{-1}(y_{lm})$.

In this procedure, we take advantage of the assumption that the sampling neighborhood is small enough and, on this neighborhood, various maps can be replaced by their linearized versions.

The maps f_i and Φ_i^v are represented as samples at vertex locations; to find a value of a map at arbitrary points of each domain U_i, we use bilinear interpolation; to invert a map we use point location and bilinear interpolation.

Finally, we note that the samples that we generate are stored in an image map. For any sample, the corresponding point in the parametric domain is computed, using point location and interpolation for texture coordinates.

When sampling texture values, some neighboring values may not have been generated yet. If some of the values are missing, then we use the nearest already-generated neighbor. If none of the values are available, then no sample is generated for this neighborhood at this location.

5 Coherent Synthesis

We now describe the *coherent synthesis* algorithm, based on Ashikhmin's algorithm for image texture synthesis [3]. This algorithm is based on the observation that the l^2-norm is an imperfect measure of perceptual similarity. Instead, it attempts to copy large coherent patches from the example texture, since coherent regions are guaranteed to have the appearance of the example texture, although the seams between patches may not. This method runs much faster than the other methods and produces higher-quality results for many textures. The multiresolution representation used in the previous section is not necessary here. However, like Ashikhmin's algorithm, this method does not work well for some smoothly varying textures.

This method runs in a single pass over all samples on the surface. For each synthesized sample, we record the floating point coordinates in the image map that this sample was copied from. To synthesize a sample x, we find nearby samples which are already synthesized and use them to look up corresponding locations in the example texture. We displace these locations in order to obtain candidate source samples for x. These candidates are chosen so that they correspond to a continuation of the samples already copied from the example. For each candidate, we collect a neighborhood of the same connectivity as the original target neighborhood, and compare it to the target neighborhood. We copy the value of the closest-matching candidate to x. Note that only a few candidates are considered, so no search acceleration is necessary. This makes it possible to resample the example texture rather than the target.

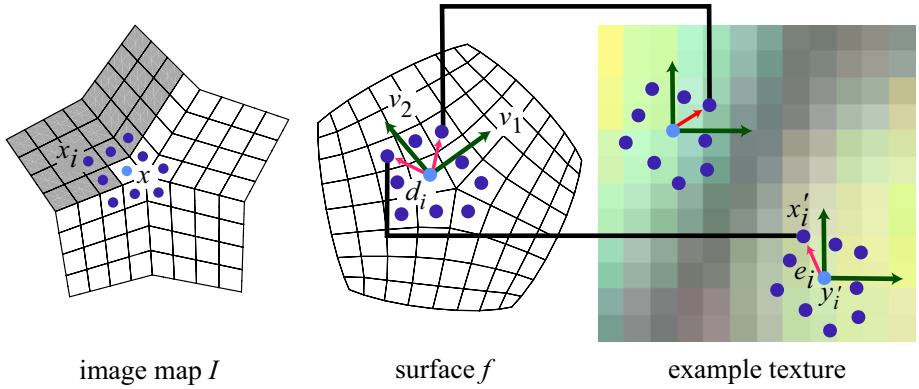


Fig. 3. Coherence synthesis. In order to synthesize a texture value for a point x, we examine each of its already-synthesized neighbors x_i. Each neighbor proposes a candidate location y'_i from the example, based on its own location in the example and its distance from x on the surface. The best candidate is computed by comparing the candidate neighborhoods with l^2.

Now we describe the algorithm in greater detail. We treat the example texture as continuous, implying that evaluation between texture samples is done by linear interpolation. The algorithm is illustrated in Figure 3.

1. To synthesize a sample x coming from an image map I, we collect all synthesized samples x_i, $i = 0 \ldots m$ in the image map I which are no more than 2 samples away from x. If x is less than two samples away from the image map boundary, we also retrieve samples in the image map sharing the boundary with I. Finally, if x is less than two samples away from a corner of an image map, we also collect adjacent samples from all image maps sharing the corner. If no synthesized samples are located a random value from the example texture is selected.
2. For each of the collected samples x_i, we compute the displacement $d_i = f(x_i) - f(x)$ of the surface locations, and project it into the tangent plane at $f(x)$ to obtain tangent displacements d_i^t. The tangent displacements can be represented in the local coordinate system (v_1, v_2) by 2D vectors e_i: $d_i^t = e_i^1 v_1 + e_i^2 v_2$.
3. Our next goal is to locate values in the example that can be potentially used to synthesize a value for x. For each of the neighboring samples x_i, we look up the corresponding location x'_i in the example texture. We use these samples to generate candidate locations $y'_i = x'_i - e_i$, i.e. by looking up the value which is located in the example in the same way with respect to x'_i as x is with respect to x_i. Note that, unlike in Ashikhmin's method, the location and the displacement should be represented as floating point numbers in order to prevent round-off error.
4. Now we need to choose which of the candidates is used to get the value for the target. To do this, we compare neighborhoods of y'_i with the neighborhood of x. We use the same set of displacements e_i, to get samples around y'_i which are arranged in the same pattern as x_i around x, i.e. we consider neighborhoods $N(y'_i)$ consisting of samples $y'_{ij} = y'_i + e_j$. Among these neighborhoods we choose the one for which the l^2 distance from the the neighborhood of x is minimal (undefined values are discarded when the distance is computed).

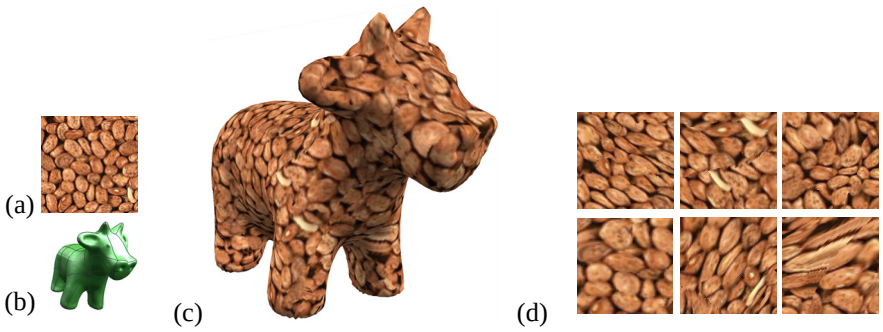


Fig. 4. Surface texture synthesis. (a) Example texture, obtained from VisTex database and downsampled to 96x96. (b) Cow model, showing edges corresponding to edges of top-level faces. (c) Synthesis result using coherent method based on Ashikhmin's algorithm [3]. (d) Representative texture maps generated during the process. Variations in surface shape appear as distortions in the texture maps. The surface is covered by a total of 50 texture maps.

6 Experiments

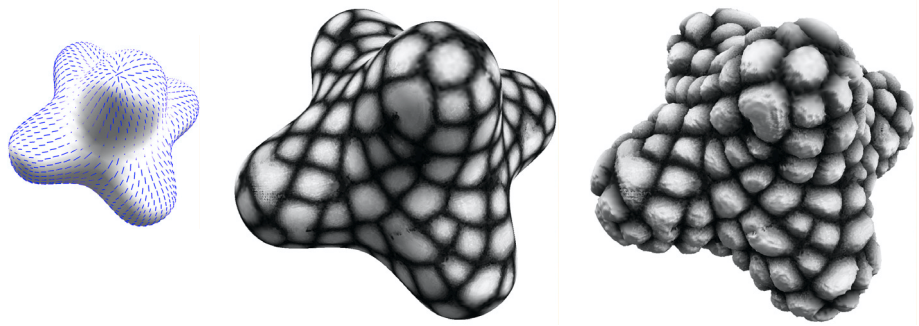
In Figure 4, we demonstrate coherent synthesis of a nut texture on a cow model. Note that the image maps appear distorted, because they are synthesized in order to appear undistorted on the surface. With coherent method, the texture has high quality, but some small discontinuities are visible. The synthesis took 168 seconds. Figures 5 and 6 show the transparency and displacement maps created with multiscale synthesis. In Figure 7, different scales of the same texture were synthesized with multiscale synthesis and the same vector field. Figure 8 compares the multiscale and coherent algorithms. The quality of the results is very similar to those of the 2D algorithms when applied to these textures. In our experience, one can predict the results of the surface synthesis by running the 2D algorithms. The green texture took 100 seconds with multiscale synthesis and 28 seconds with coherent synthesis. The flower texture took almost five hours with multiscale synthesis and two minutes with coherent synthesis. Figure 9 demonstrates synthesis of transparency maps. The results of coherent synthesis on complex models are displayed in Figure 10. All the timings are from a 550 MHz Pentium III.

7 Discussion and Future Work

We have presented efficient methods for synthesizing a texture onto a 3D surface from a 2D example texture. For example, these methods produce textures with similar quality and speed to their 2D counterparts. This means that those textures that work well with Ashikhmin's algorithm work well with our coherence algorithm. Hence, there is reason to hope that these strategies may be applied when new 2D texture synthesis algorithms are developed.

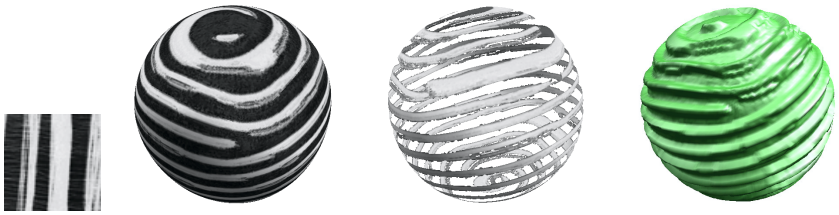
There are a number of directions for future work. All of our sampling operations use either point sampling or bilinear sampling. However, since texels represent area integrals of a function, full area integrals should be used instead. We have designed our methods to avoid some sampling artifacts while maintaining efficiency; however, more work remains to be done in order to study and improve the theoretical properties.

Surface reflectance functions and material properties, such as BRDFs or fur [12], can be synthesized, perhaps via straightforward extensions of the ideas presented here.



(a) (b) (c)

Fig. 5. (a) Surface with orientation field. Note the singularity of the field at the top of the model. (b) Synthesized texture using multiscale synthesis of the first texture from Figure 8 in grayscale. The texture appears consistent at the singularity. (c) Texture mapping and displacement mapping with the same texture.



(a) (b) (c) (d)

Fig. 6. Textured sphere generated with multiscale synthesis. (a) Example texture. (b) Synthesized texture. (c) Texture mapping plus transparency mapping (using the same texture). (d) Displacement mapping.

Fig. 7. Zebra dog, generated with multiscale synthesis. Varying the scale parameter creates a texture with a different size on the surface.

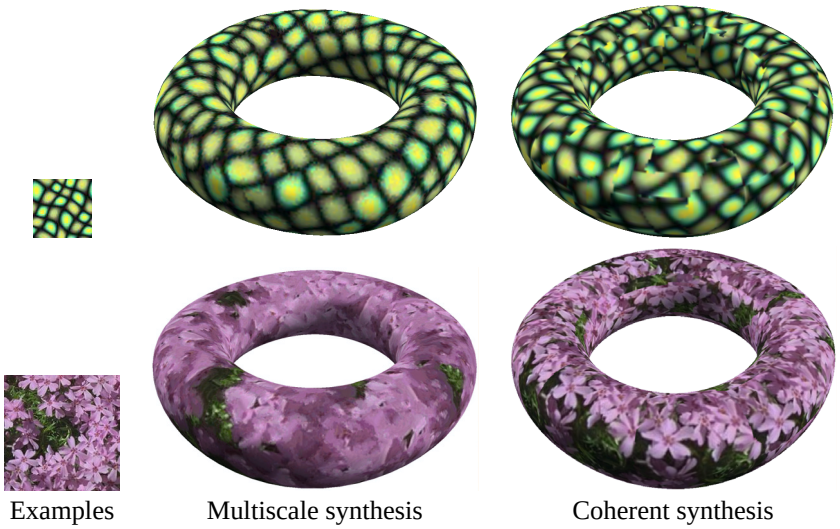


Fig. 8. Comparison of multiscale (based on Wei-Levoy [18]) and coherent algorithms (based on Ashikhmin [3]).

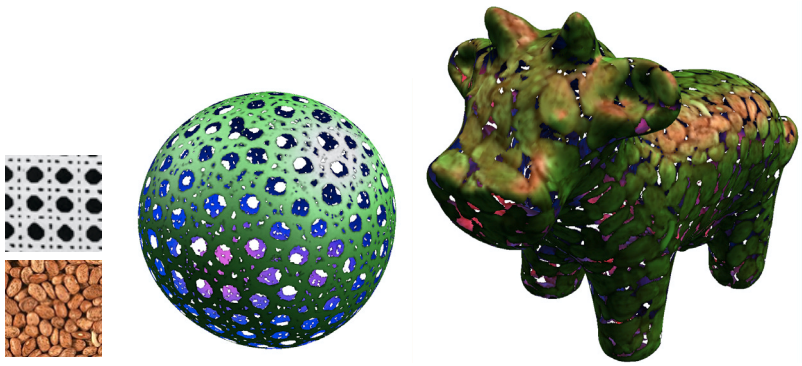


Fig. 9. Transparency mapping. *Left:* Example textures. *Middle:* Wicker ball, generated from first texture by multiscale synthesis. *Right:* Bronze cow, generated by coherent synthesis from second texture blended with a green surface color.

Fig. 10. Chia cow and sea horse, generated with coherent synthesis.

Real-time procedural shaders that perform example-based texture synthesis would allow complex surface textures to be generated in real-time. However, this appears difficult because the current texture synthesis algorithms require samples to be generated sequentially.

References

1. Marc Alexa, Daniel Cohen-Or, and David Levin. As-Rigid-As-Possible Shape Interpolation. *Proceedings of SIGGRAPH 2000*, pages 157–164, July 2000.
2. Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. *Journal of the ACM*, 45(6):891–923, 1998. Source code available from http://www.cs.umd.edu/~mount/ANN.
3. Michael Ashikhmin. Synthesizing natural textures. *2001 ACM Symposium on Interactive 3D Graphics*, pages 217–226, March 2001.
4. Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. *Proceedings of SIGGRAPH 97*, pages 361–368, August 1997.
5. Alexei Efros and Thomas Leung. Texture Synthesis by Non-parametric Sampling. *7th IEEE International Conference on Computer Vision*, 1999.
6. Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.
7. Cindy M. Grimm and John F. Hughes. Modeling surfaces of arbitrary topology using manifolds. *Proceedings of SIGGRAPH 95*, pages 359–368, August 1995.
8. Igor Guskov, Kiril Vidimce, Wim Sweldens, and Peter Schrder. Normal meshes. *Proceedings of SIGGRAPH 2000*, pages 95–102, July 2000.
9. David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. *Proceedings of SIGGRAPH 95*, pages 229–238, August 1995.
10. Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. *Proceedings of SIGGRAPH 2000*, pages 517–526, July 2000.
11. Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. *Proceedings of SIGGRAPH 98*, pages 95–104, July 1998.
12. Jerome E. Lengyel, Emil Praun, Adam Finkelstein, and Hugues Hoppe. Real-time fur over arbitrary surfaces. *2001 ACM Symposium on Interactive 3D Graphics*, pages 227–232, March 2001.
13. Fabrice Neyret and Marie-Paule Cani. Pattern-based texturing revisited. *Proceedings of SIGGRAPH 99*, pages 235–242, August 1999.
14. J. Portilla and E. P. Simoncelli. A Parametric Texture Model based on Joint Statistics of Complex Wavelet Coefficients. *International Journal of Computer Vision*, 40(1):49–71, December 2000.
15. Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. *Proceedings of SIGGRAPH 2000*, pages 465–470, July 2000.
16. Peter-Pike J. Sloan, III Charles F. Rose, and Michael F. Cohen. Shape by example. *2001 ACM Symposium on Interactive 3D Graphics*, pages 135–144, March 2001.
17. Greg Turk and James O'Brien. Shape transformation using variational implicit functions. *Proceedings of SIGGRAPH 99*, pages 335–342, August 1999.
18. Li-Yi Wei and Marc Levoy. Fast Texture Synthesis Using Tree-Structured Vector Quantization. *Proceedings of SIGGRAPH 2000*, pages 479–488, July 2000.
19. Song Chun Zhu, Ying Nian Wu, and David Mumford. Filters, Random fields, And Maximum Entropy: Towards a Unified Theory for Texture Modeling. *International Journal of Computer Vision*, 12(2):1–20, March/April 1998.
20. D. Zorin, P. Schröder, T. DeRose, A. Levin L. Kobbelt, and W. Sweldens. Subdivision for modeling and animation. SIGGRAPH 2000 Course Notes.

Progressive Geometry Compression

Andrei Khodakovsky
Caltech

Peter Schröder
Caltech

Wim Sweldens
Bell Labs

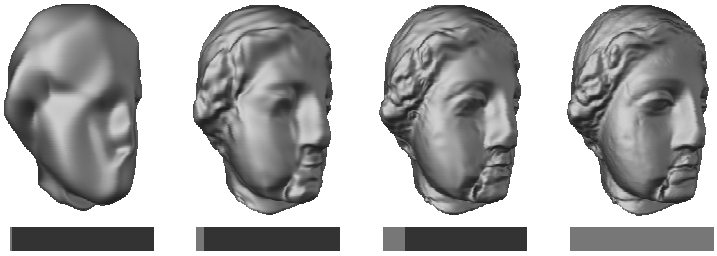
Motivation

- 3D scanning
  - densely sampled, highly detailed geometry
  - up to billions of samples
  - efficient storage and transmission critical

Goals

Geometry compression

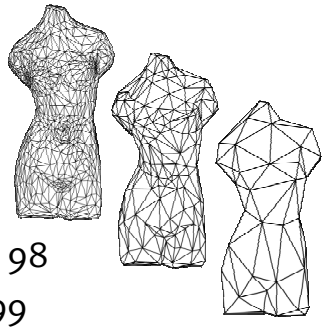
- methods for “large” geometry
- progressivity critical



Previous Work

Mesh = connectivity + vertices

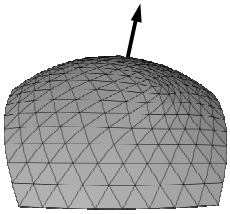
- connectivity: 2b/v
- vertices
  - single: Touma et al. 98
  - progressive:
    - MPEG4: Taubin et al. 98
    - CPM: Pajarola et al. 99
    - Cohen-Or et al. 99



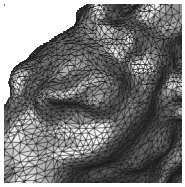
Our Key Observations

What is in a mesh?

connectivity + vertices

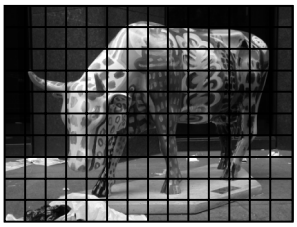


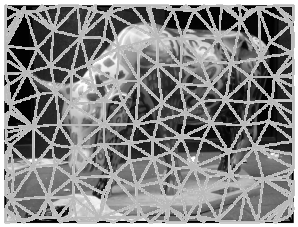
sample locations



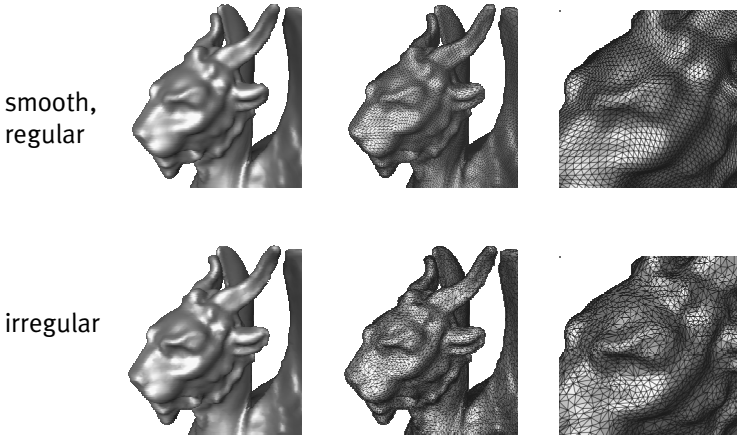
geometry

Image

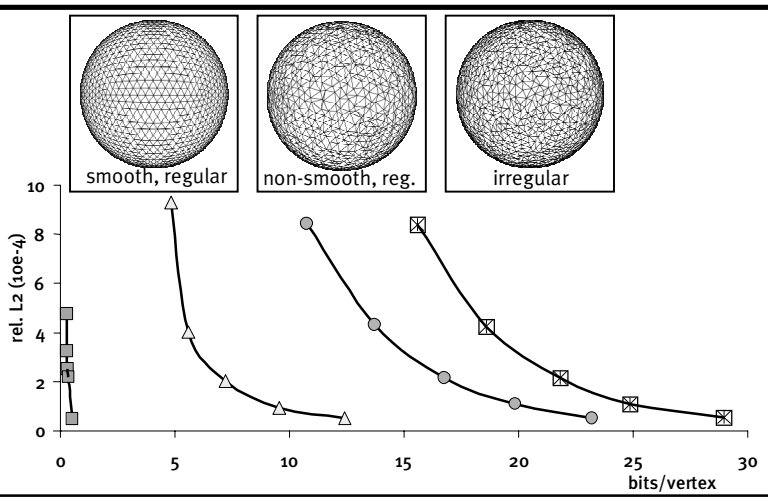




3D Surface



An Example

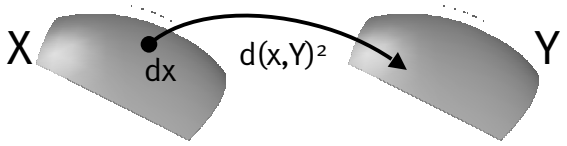


Care About Geometry!

Best geometry for given rate

- change parameterization
- geometric error: L_2 average

$$d_{NS}^2(X,Y) = \frac{1}{|X|}\int d(x,Y)^2 \, dx$$



Our Approach

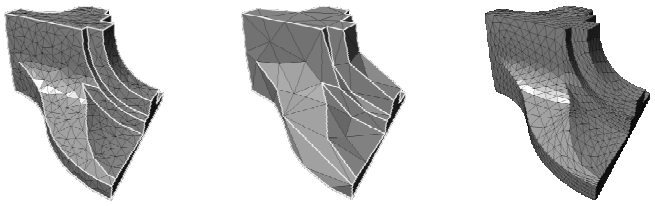
Get rid of parameter and connectivity information!

- rate-distortion performance
- resample onto semi-regular mesh
- wavelet transform
- zero-tree coding

RESAMPLE

Build semi-regular representation

- MAPS algorithm [Lee et al. 1998]
  - remeshing induces error
    - comparable with discretization error



WAVELETS

Fundamental mathematical tool

- representation
  - objects and operators
- algorithms
  - fast transform
  - speed/accuracy tradeoff
  - simple implementation

Wavelets

Why is this connection useful?

- wavelets exploit coherence
  - split signal into high and low pass
  - high pass much cheaper to encode
  - "most coefficients small most of the time"
- example: image compression
- how to apply to geometry?

Wavelet Connection

Scaling functions via subdivision

- pure subdivision has all wavelet coefficients equal to zero
- what's missing are the details (high frequencies)

What we gained

- wavelet machinery in arbitrary topology setting!

Constructing Wavelets

Traditional

- Fourier tools
  - dilation/translation setting

Here:

- refineable function setting only
- need tool in spatial domain

Lifting Scheme

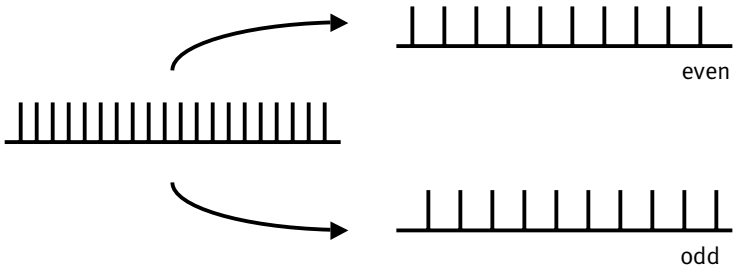
General wavelet construction tool

- 3 stages
  - lazy wavelet: split data
  - predict: compute detail
  - update: compute smooth approximation
- algebra only...
  - analysis still needs to be performed

Lazy Wavelet

Subsampling

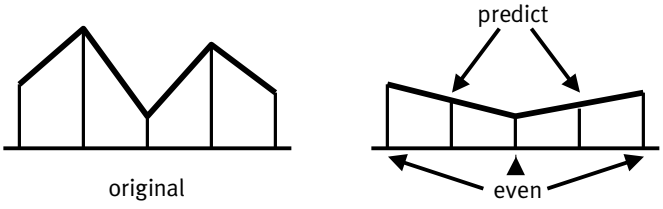
- split into even and odd samples



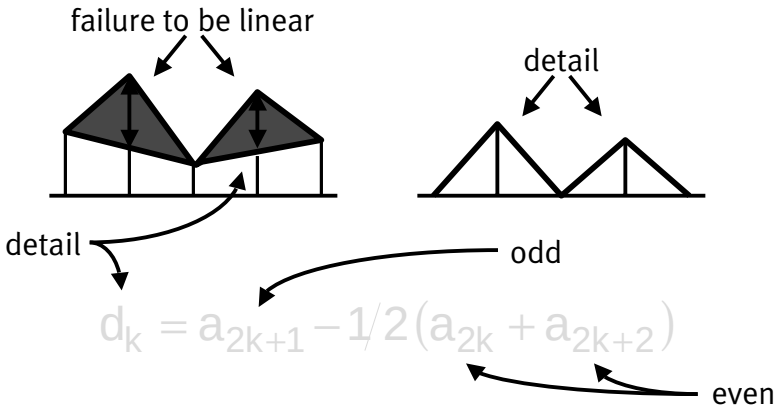
Prediction

Use even to predict odd

- keep difference with prediction
- detail = odd - predict(even)



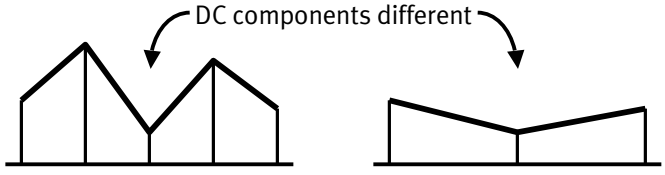
Prediction



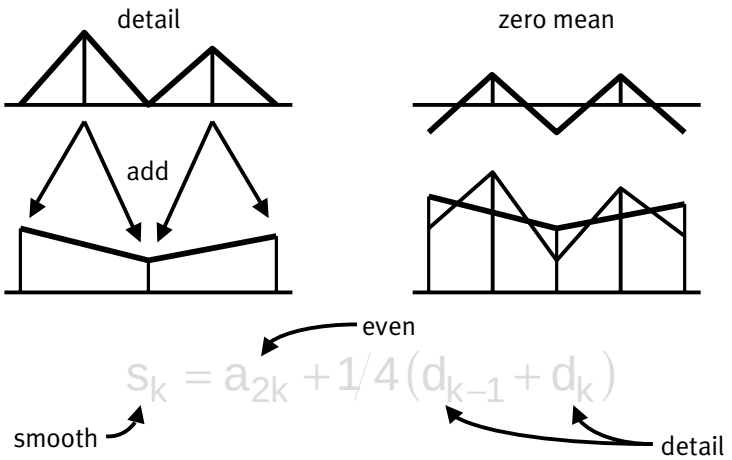
Update

Even values are subsampled

- aliasing!
- smooth = even + update(detail)



Update



Lifting

General technique

- repeated lifting allows construction of all classical wavelets with finite support
  - basis of JPEG2000
- carries over into arbitrary topology setting
  - for example: Geometry!

Decomposition

Subdivision

- prediction operator in lifting setting
- differences tend to be small
  - better encoding
- hierarchical setting for progressive transmission
  - zero tree coders

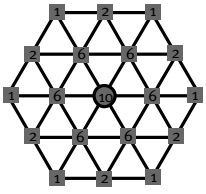
Loop Wavelets

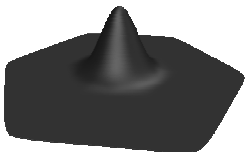
Refinement relations

- subdivision yields scaling functions

$$p^{j+1} = Sp^j$$

$$\varphi(t) = \sum_{k \in Z^2} s_k \varphi(2t - k)$$





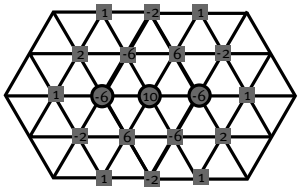
scaling function

- primal scaling function fixed
  - find completion

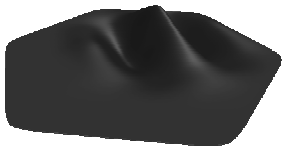
Completion

Completion yields wavelet filters

- want finite reconstruction filters
- QMF for Q, but: $(SQ)^{-1}$ not finite



$$\psi(t) = \sum_{k \in Z^2} d_k \varphi(2t - k)$$

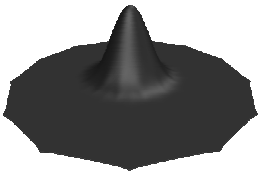


wavelet
1 of 3 different orientations

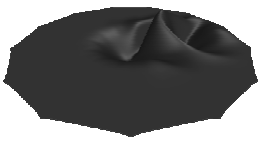
Loop Wavelets

Irregular vertices

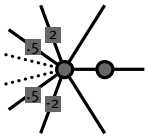
- split filter around irregular vertex
  - results in good conditioning
  - boundaries: extension



Scaling function, valence 13



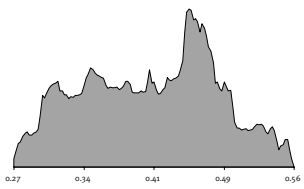
Wavelet function, valence 13



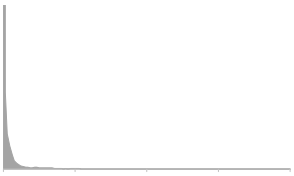
Wavelet Transform

Effect of wavelet transform

- changes distribution of coefficients
  - almost all coefficients close to zero



Vertex coordinates

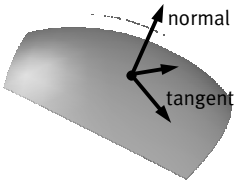


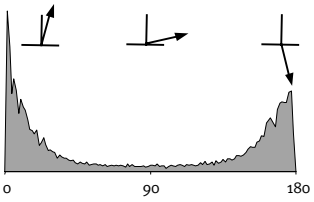
Wavelet coefficients

Local Frames

Coefficients are vector valued

- normal direction most important!!





- best results with scalar quantization
- finer quantization of normal direction

PROGRESSIVE ENCODING

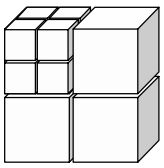
Make each bit count

- no sense in sending approximate geometry very precisely…
- sending coefficients one at a time not enough

ENCODING

Different options

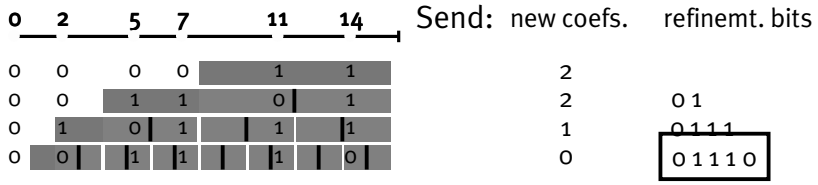
- coordinates: polar, cartesian
- quantization: vector, scalar
  - vector does not pay off
  - threshold
  - enumerate cell name
  - Huffman coding



BITPLANE ENCODING

Progressive compression:

- encode largest coefficients first
  - encode only significance bit
  - subsequent bits in later iterations

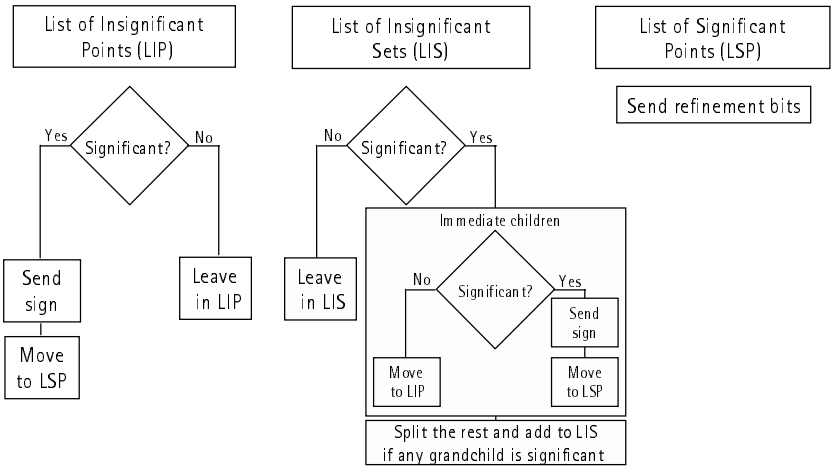


ZEROTREE

Why it works

- be smart about the location of large coefficients
  - small parents tend to have smaller children ⇢ entropy coding gain
- run state machine at encoder and decoder
  - only send synchronization bits

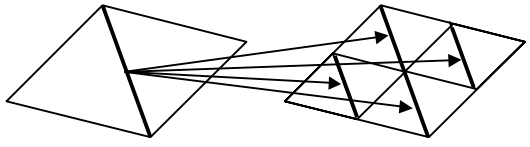
ZEROTREE ALGORITHM



ZERO-TREE

Need tree structure for coefficients

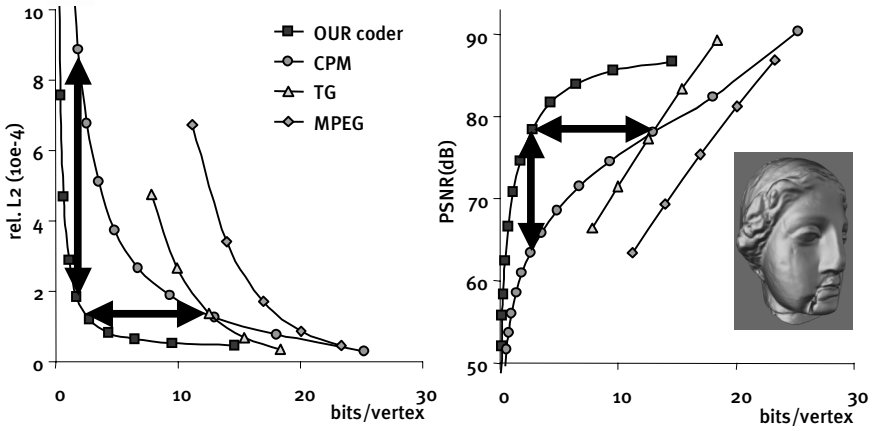
- wavelets live on edges



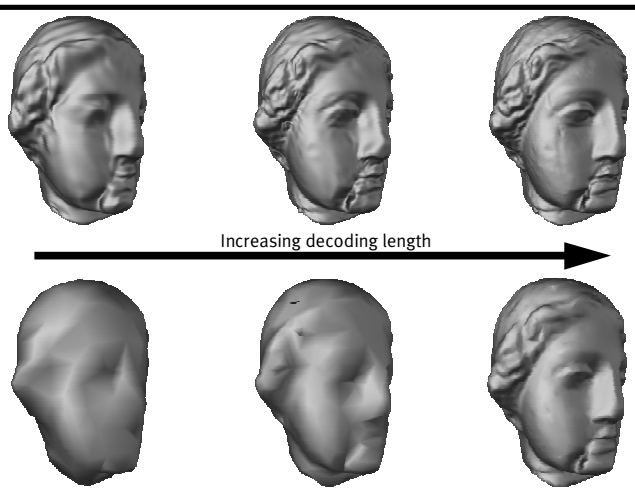
Test whole tree for significance

- split tree isolating significant coefs

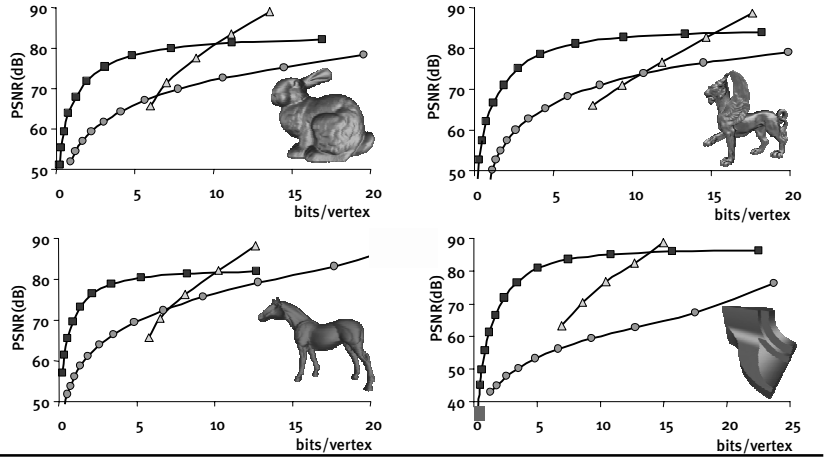
Results I



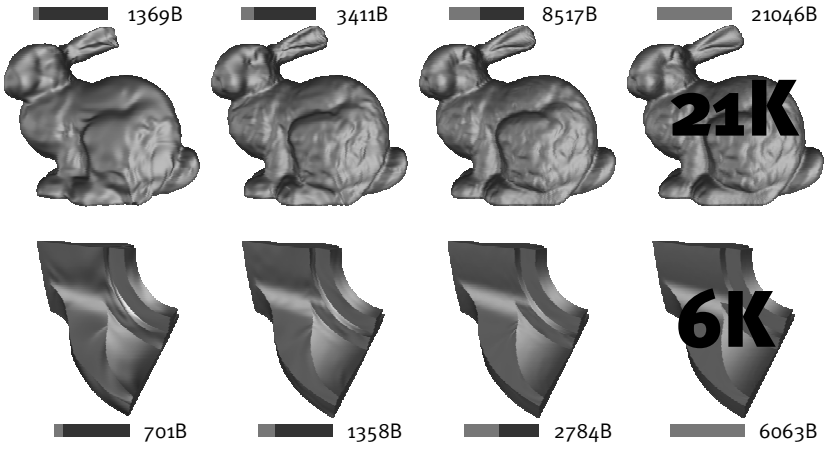
Visual Comparison



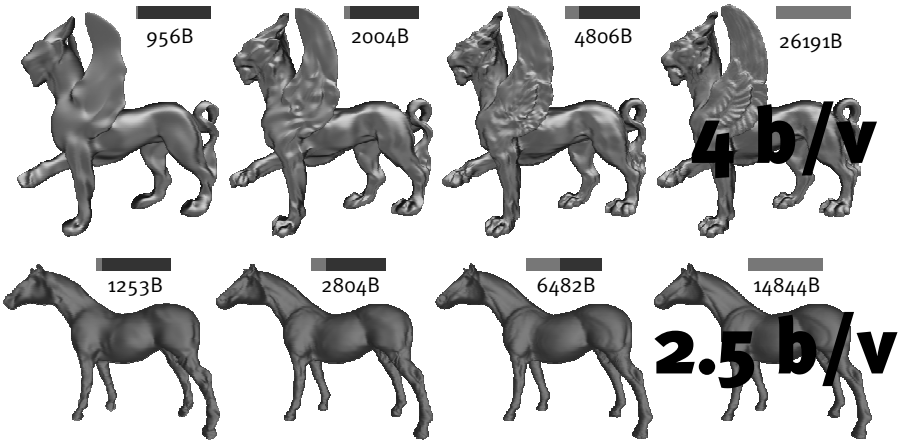
Results II



Results III



Results IV



More Info

Resources

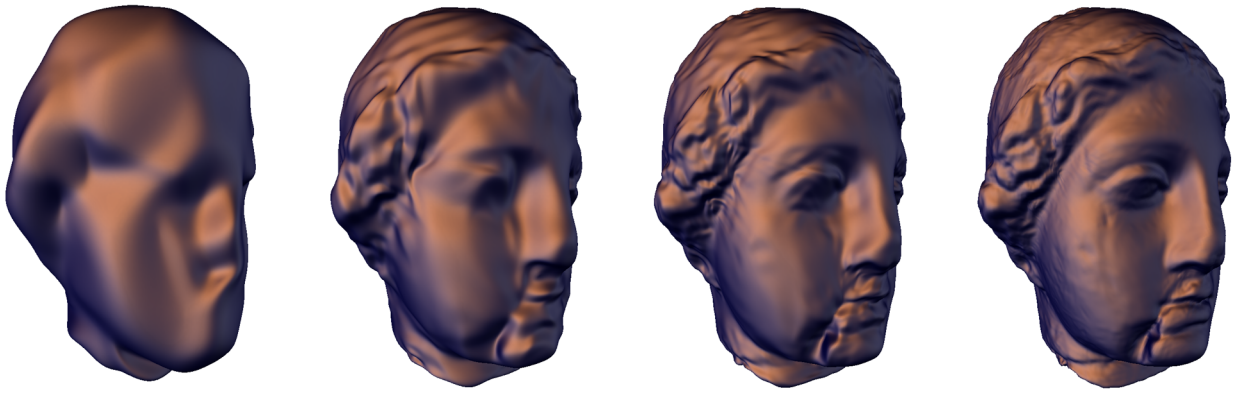
- examples and executable
  - http://multires.caltech.edu/software/pgc
- zerotree code
  - http://www.cipr.rpi.edu/research/SPIHT/
  - http://www.cs.dartmouth.edu/~gdavis/wavelet/wavelet.html
- questions
  - Andrei Khodakovsky, akh@cs.caltech.edu
  - Igor Guskov, ivguskov@cs.caltech.edu

Progressive Geometry Compression

Andrei Khodakovsky
Caltech

Peter Schröder
Caltech

Wim Sweldens
Bell Laboratories



476B (e:40) 1528B (e: 12) 4163B (e: 4.7) 26800B (e: 0.82)

Figure 1: *Partial bit-stream reconstructions from a progressive encoding of the Venus head model. File sizes are given in bytes and relative L^2 reconstruction error in multiples of 10^{-4}. The rightmost reconstruction is indistinguishable from the original.*

Abstract

We propose a new progressive compression scheme for arbitrary topology, highly detailed and densely sampled meshes arising from geometry scanning. We observe that meshes consist of three distinct components: geometry, parameter, and connectivity information. The latter two do not contribute to the reduction of error in a compression setting. Using semi-regular meshes, parameter and connectivity information can be virtually eliminated. Coupled with semi-regular wavelet transforms, zerotree coding, and subdivision based reconstruction we see improvements in error by a factor four (12dB) compared to other progressive coding schemes.

CR Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling - *hierarchy and geometric transformations*; G.1.2 [**Numerical Analysis**]: Approximation - *approximation of surfaces and contours, wavelets and fractals*; I.4.2 [**Image Processing and Computer Vision**]: Compression (Coding) - *Approximate methods*

Additional Keywords: Compression algorithms, signal processing, wavelets, subdivision surfaces, semi-regular meshes, zerotree coding, hierarchical representations

1 Introduction

Today we can accurately acquire finely detailed, arbitrary topology surfaces with millions and most recently billions [22] of vertices. Such models place large strains on computation, storage, transmission, and display resources. Compression is essential in these settings and in particular *progressive* compression, where an early, coarse approximation can subsequently be improved through additional bits. While compression of *images* has a long history and has achieved a high level of sophistication, compression of *surfaces* is relatively new and still evolving rapidly.

Compression is always a tradeoff between accuracy and bit rate, i.e., bits per vertex. This tradeoff is the subject of classical rate-distortion theory. While rate-distortion curves are common in the image coding literature they have only recently appeared in geometry coding. This is partially due to the fact that the error for images is easily measured using the L^2 norm of the difference between original and approximation, while measuring error for surfaces is more involved. Since there is no immediate correspondence between the original and compressed surface, one cannot simply subtract one surface from another. This difficulty is typically addressed by computing a geometry error using, for example, Hausdorff distance. Such error metrics do not depend on the particular sample locations or connectivity, but instead measure the distance between the geometric shapes. This is important since the original and compressed mesh may have very different sample locations and connectivity, especially in a progressive setting. By sample location we mean the precise location of the vertex *within* the surface.

How low can such errors be? Consider a continuous physical surface, such as the Venus sculpture whose scan generated the mesh in Figure 1. Given that the source geometry is continuous, any digital representation, such as a triangle mesh, has some error E associated with it. This error has three components due to sampling, discretization, and quantization. Sampling error E_s arises from acquisition noise. Discretization error E_d is due to the fact that a triangulation with edge length h can approximate a smooth geometry no better than $O(h^2)$. Finally, a finite bit representation for the vertex positions leads to quantization error E_q. The sampling and triangulation of the model fix E_s and E_d. A standard float representation typically leads to a quantization error much smaller than $E_s + E_d$. All existing single rate coders proceed by first quantizing the vertex positions more coarsely leading to a quantization error $E'_q \approx E_s + E_d$ followed by lossless encoding of the connectivity and quantized vertex positions. Existing progressive coders aim

to eventually recover the quantized sample locations and original connectivity. For small meshes with carefully layed out connectivity and sample locations this is very appropriate. The situation is different for highly detailed, densely sampled meshes coming from 3D scanning: Since distortion is measured as geometric distance the sample locations and connectivity can be treated as additional degrees of freedom to improve the rate-distortion performance. As long as the final result has geometric error on the order of the original E, the actual sample locations and connectivity do not matter. We will call the information contained in the sample locations, the *parameter* information. For example, by letting the vertices slide *within* the surface we only change the parameter information and not the geometric fidelity.

In particular, we propose a new progressive geometry compression method which is based on smooth semi-regular meshes, i.e., meshes built by successive triangle quadrisection starting from a coarse irregular mesh. Almost all vertices in a semi-regular mesh have valence six and their sample locations can easily be estimated. Hence, semi-regular meshes allow us to eliminate almost all *parameter* and connectivity information. As we illustrate below, parameter and connectivity information make up a considerable fraction of the bit budget in existing coders, but do not contribute at all to reducing geometric error. Consequently our rate-distortion curves are significantly better than those of existing coders. For most models, our error is about four times smaller at comparable bit rates, a remarkable 12 dB improvement!

Semi-regular meshes additionally allow for wavelet transforms and zerotree coders. Zerotrees are amongst the best image coding algorithms today. Wavelets have superior decorrelation properties and allow for subdivision based reconstruction. This means that in regions where the encoder sets wavelet coefficients to zero the decoder uses subdivision to reconstruct the geometry. Hence even highly compressed surfaces are still smooth and visually pleasing. Figure 1 shows a sequence of progressive reconstructions of the compressed Venus model at different bitrates.

Goals and Contributions The main contribution of this paper is the observation that parameter information makes up a significant fraction of the bit budget while not contributing to error reduction at all. This motivates our compression algorithm based on semi-regular meshes.

As input our algorithm takes an irregular mesh describing a 2-manifold (possibly with boundary) and produces successive approximations employing semi-regular meshes with little parameter and connectivity information. The coder first produces a hierarchical approximation of the surface which is subsequently encoded with a zerotree progressive coder. Novel aspects of the algorithm include

- reducing parameter information through the use of semi-regular meshes;
- a Loop based wavelet transform for high order decorrelation and subdivision based reconstruction;
- a novel zerotree hierarchy for primal semi-regular triangle meshes of arbitrary topology.

We emphasize that our target application is the compression of densely sampled, highly detailed surfaces. Our algorithm is not effective when the input geometry is well described by a small, carefully layed out mesh. In this case progressive coding is generally questionable and non-progressive coders are more appropriate and perform exceedingly well.

1.1 Review of Related Work

Mesh Compression: Algorithms for efficient encoding of arbitrary connectivity meshes have been described both for the progressive and non-progressive setting (for an excellent overview of 3D geometry compression see [36]). Most of the early efforts concentrated on finding efficient encodings for mesh connectivity with the current state of the art at around 2-6b/v (bits per vertex) [37, 13, 35, 29, 28]. Vertex positions are dealt with by performing an initial quantization followed by predictive coding induced by the traversal order of the connectivity encoding.

In contrast to single target rate coders, progressive coders aim to code for a range of rates by allowing reconstruction of intermediate shapes using a prefix of the encoded bit stream. Such coding schemes are typically based on mesh simplification techniques. Examples include progressive meshes [26, 23, 16], independent set vertex removal strategies [4], topological surgery [34], and topological layering [1]. Connectivity bits increase to around 4-10b/v in these schemes. Prediction of vertex positions is now more naturally performed in a hierarchical fashion as induced by the associated mesh simplification. Examples include centroid predictors [34, 4] as well as higher order predictors [26]. To date, progressivity in these coders has typically been focused on connectivity encoding. Rate-distortion theory however says that coordinate values should be progressively quantized [23, 17] as well: to minimize error at a given rate one must trade off additional quantization bits for already present vertices against bits for new vertices and their connectivity.

Wavelets It is well known from image coding that wavelet representations are very effective in decorrelating the original data [8, 6], greatly facilitating subsequent entropy coding. In essence, coarser level data provides excellent predictors for finer level data, leaving only generally small prediction residuals for the coding step. For tensor product surfaces many of these ideas can be applied in a straightforward fashion [8, 33, 12]. However, the arbitrary topology surface case is much more challenging. To begin with, wavelet decompositions of general surfaces were not known until the pioneering work in [25]. These constructions were subsequently applied to progressive approximation of surfaces [2] as well as data *on* surfaces [31, 19].

Multiresolution surface representations based on subdivision [39] and local frame details are closely related to our wavelet constructions and have proven to be very powerful in a variety of circumstances. However, they require the initial surface to be represented by a semi-regular mesh. This has led to the development of a number of algorithms for remeshing [10, 20, 21, 18].

Zerotree Coders Some of the best wavelet based progressive coders are based on zerotrees [5, 32, 30]. They effectively exploit the fact that wavelet coefficients at finer scales tend to be smaller in magnitude than coefficients at coarser scales in the same region. A zerotree coder encodes the location of coefficients below threshold in subtrees. Standard zerotree coders for images are based on a dual formulation, i.e., coefficients are associated with faces. For primal hierarchical mesh decompositions using face splits (e.g., quadrisection of triangles) the data however lives at vertices, not faces. We show in Section 3.4 how to build zerotree coders for primal hierarchies.

Irregular Subdivision Our separation of parameter versus geometry information is partially inspired by the work done on irregular subdivision [14] and intrinsic curvature normal flow [7]. They point out that without the parameter side information, it is impossible to build high order schemes converging to smooth meshes. Irregular parameter information is inherently hard to encode and hinders the performance of irregular mesh coders.

2 Geometry, Parameter, and Connectivity Information

Elimination of parameter and connectivity information is a key ingredient of our algorithm. In this section we go into more detail

regarding parameter and connectivity information and how to eliminate it.

Previous compression approaches have typically treated triangle meshes as consisting of *two* distinct components: connectivity and vertex positions. State of the art coders are able to encode connectivity of irregular meshes with 2b/v or even less. Hence, it is argued, vertex positions are much more expensive and their coding needs further advancement, for example through better predictors.

The main insight of this paper is that there are actually *three* components: connectivity, geometry, and *parameter* information. The parameter information captures where the sample locations are *within* the surface while the geometry information captures the geometry *independent* of the sample locations used. So far parameter and geometry information were treated together.

Consider a vertex of a particular Venus head triangulation. Moving this vertex slightly *within* the surface, does not change the discretization error or geometry information. It only affects the parameter information. Alternatively, moving the vertex normal to the surface clearly changes the error and geometry information, but leaves parameter information unchanged. This illustrates that while geometry and parameter information are globally intertwined they disconnect locally: infinitesimally, we may think of parameter information as being described by displacements in the tangent plane to the surface. Geometry information on the other hand is normal to the surface. This implies that from a rate distortion point of view bits should be allocated preferentially to the local normal direction. For smooth parameterizations this occurs naturally since prediction residuals in the tangent plane will be small.

Sphere Example To illustrate the power of the distinction between geometry, parameter, and connectivity information we consider three triangulations of a sphere (Figure 2). All three meshes contain the same geometry information and carry the same discretization error E_d with no sampling noise. The first two meshes have semi-regular connectivity but different parameter information. The middle one was generated by jiggling the sample locations within the sphere, thereby adding significant parameter information. The rightmost has irregular connectivity and parameter information.

Figure 3 shows the respective rate-distortion curves when using the state of the art non-progressive coder of Touma and Gotsman (TG) [37]. We always show non-progressive curves dashed since these points are not achievable in a progressive manner. In case of the smooth semi-regular mesh, the TG coder correctly noticed that it contains almost no connectivity information (0.1 b/v) and almost no parameter information. Its performance is essentially limited by the quality of the predictor used. The TG coder for the non-smooth semi-regular sphere is worse illustrating the bit penalty for parameter information. The TG coder for the irregular mesh (right) illustrates the additional overhead from irregular connectivity. This example demonstrates the tremendous pay off of reducing both connectivity and parameter information in a mesh.

Finally the small curve near the y-axis shows the result of applying our coder to the smooth semi-regular mesh. It can approximate the sphere with a relative error of $5 \cdot 10^{-5}$ using 166 bytes or .5 b/v. This it not surprising since a sphere has very little geometric information and a smooth semi-regular mesh is essentially optimal for our coder. This is where the high order decorrelation and subdivision based reconstruction really pays off. The same effect we see here so pronounced for the sphere, can also be observed in smooth, regularly sampled regions of more general surfaces, see Section 4.

3 Algorithm Components

The algorithm accepts as input an arbitrary connectivity 2-manifold (with boundary) triangulation. In a first step we compute a smooth

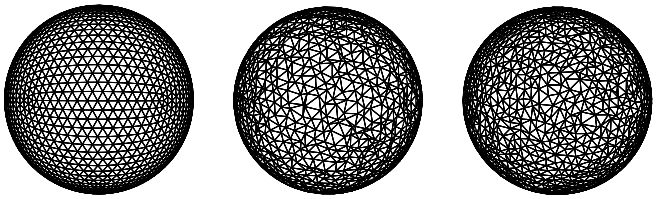


Figure 2: *Three spherical meshes each with 2562 vertices: smooth semi-regular (left), non-smooth semi-regular (middle), irregular (right). They have the same geometry information. The middle one also has parameter information while the right one has parameter and connectivity information.*

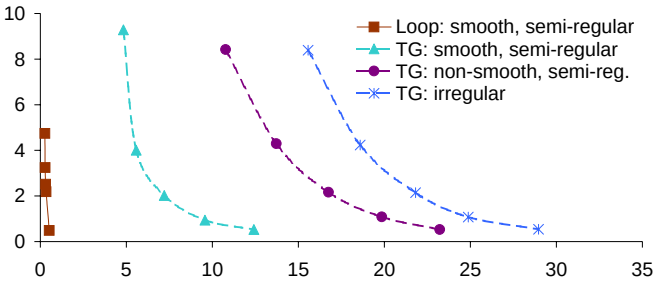


Figure 3: *Rate distortion curves for the triangle meshes from Figure 2 measured in relative L^2 error on a scale of 10^{-4} as a function of rate in b/v for TG coordinate quantization levels of $8-12b$.*

global parameterization using the MAPS algorithm [21]. This allows us to compute successive adaptive approximations with semi-regular connectivity. These semi-regular approximations are subsequently wavelet transformed and progressively compressed using zerotrees. The coarsest level connectivity is encoded using a standard non-progressive mesh encoder [37]. The decoder may produce intermediate approximations from any prefix of the bitstream.

We need to define the distance $d(X,Y)$ between two surfaces X and Y. Let $d(x,Y)$ be the Euclidean distance from a point x on X to the closest point on Y. Then the L^2 distance $d(X,Y)$ is given by

$$d(X,Y) = \left(\frac{1}{\text{area}(X)} \int_{x \in X} d(x,Y)^2 dx \right)^{1/2}.$$

This distance is not symmetric and we symmetrized it by taking the max of $d(X,Y)$ and $d(Y,X)$. For triangulations this distance can be computed using the METRO tool [3]. All the L^2 errors reported here are relative with respect to the bounding box diagonal on a scale of 10^{-4}, while rate is reported in b/v with respect to the number of vertices in the original input mesh.

3.1 Parameterization

As a first step, we compute a smooth parameterization of our input triangulation using MAPS [21]. An important feature of MAPS is its ability to automatically align iso-parameter lines of the semi-regular mesh with sharp features of the original input surface helping to avoid large wavelet coefficients near creases.

MAPS builds a bijective map between the input mesh T and a coarse base domain B. One can then apply quadrisection in the base domain B and use the mapping to build semi-regular approximations of T. These approximations have some remeshing error E_r with respect to T. While this error can be made arbitrarily small, it does not make sense to make the remeshing error E_r smaller than the discretization error E_d. This roughly occurs when the triangles from the semi-regular mesh are about the same size as the triangles of the input mesh. Using smaller triangles only serves to produce a better approximation of the input mesh, not necessarily of the original unknown geometry.

Of course one does not know E_d. An order estimate of E_d can be computed by measuring the distance between the input mesh T and a much finer mesh S obtained by Butterfly subdividing T. The latter serves as a proxy for the unknown original geometry. Once our semi-regular mesh error E_r is below the estimated discretization error E_d there is no need to further refine the semi-regular mesh. Hence our rate distortion curves will asymptotically not go to zero, but converge to the E_d estimate. Table 1 gives the E_d estimate, the minimum remeshing error, and the connectivity coding cost in bytes of the base domain B for various models. The connectivity was encoded using the TG coder.

	Feline	Bunny	Horse	Venus	Fandisk
# Vert.	49864	34835	48485	50002	6475
E_d (10^{-5})	7.3	9.4	6.0	5.5	28
E_r (10^{-5})	6.3	7.4	5.1	4.2	4.8
# Base Vert.	250	127	112	196	73
Base conn. (B)	122	76	62	72	46

Table 1: *Statistics for example meshes.*

3.2 Wavelet Transform

The wavelet transform replaces the original mesh with a coarsest mesh and a sequence of wavelet coefficients expressing the difference between successive levels. Since we deal with piecewise smooth models, neighboring vertices are highly correlated. The wavelet transform removes a large amount of this correlation. The distribution of wavelet coefficients is centered around zero and their magnitude decays at finer levels with the rate of decay related to the smoothness of the original surface. This behavior of the magnitude of wavelet coefficients is the key to progressive coding and justifies the choice of the zerotree coder for the bit encoding of coefficients.

Several methods for building wavelet transforms on semi-regular meshes exist [25, 31]. These are typically based on interpolating subdivision schemes such as Butterfly [9, 38]. A detailed description of the construction of lifted Butterfly wavelets can be found in [31]. The advantage of lifted wavelets is that both forward and inverse transforms can be computed with finite filters.

We use a novel Loop [24] wavelet transform, which has the advantage that the inverse transform uses Loop subdivision. Experimentally, we found it has rate distortion curves essentially identical to Butterfly, but typically better visual appearance.

The choice of Loop subdivision fixes the low pass reconstruction filter $\mathbf{P}$ in a wavelet construction. We require a high pass reconstruction filter $\mathbf{Q}$. Together they define the inverse wavelet transform

$$\mathbf{p}^{j+1} = \begin{bmatrix} \mathbf{P} & \mathbf{Q} \end{bmatrix} \begin{bmatrix} \mathbf{p}^j \\ \mathbf{d}^j \end{bmatrix}, \qquad (1)$$

where $\mathbf{p}^j$ are the usual control points and $\mathbf{d}^j$ the wavelet coefficients at level j. For performance reasons we would like $\mathbf{Q}$ to have small support. One way to achieve this is to apply a quadrature mirror construction [27], deriving a high pass from a low pass filter. The result is shown in the regular case in Figure 4. Note that a globally consistent choice of the sign-flipping direction is possible only for orientable surfaces. Though we can use the same stencils in the general case, the wavelet subbands corresponding to edges of a certain orientation are well-defined only for orientable surfaces.

Around irregular vertices $\mathbf{P}$ is modified as usual. For edges immediately adjacent to an irregular vertex, $\mathbf{Q}$ must be modified as well. The only taps of the $\mathbf{Q}$ filter that can fall onto irregular vertices are the two -6 coefficients left and right of the center. If one of them is irregular we essentially "open up" that part of the filter and parameterize the coefficients by edge number, counting from the "10" (Figure 4, right). If an irregular vertex has valence less than six this leads to the stencil folding over on itself, while for valences larger than six a gap is left. There is currently no theory available for wavelet constructions around irregular vertices. The only justification of the "trick" we used is that it does not impact the numerically computed condition numbers of our transform. Finally, boundaries are dealt with in the usual way through reflection.

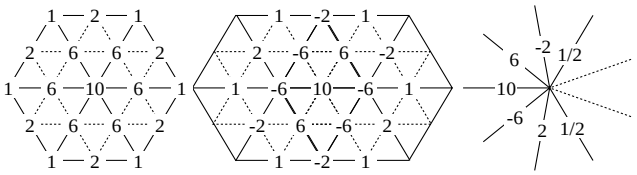


Figure 4: *Low (left) and high (middle) pass Loop reconstruction filters in the regular case. For irregular vertices the high pass filter is opened as indicated on the right.*

The forward wavelet transform, which goes from finer to coarser levels, is defined as the solution $[\mathbf{p}^j, \mathbf{d}^j]$ of the linear system in Eq. 1 for a given $\mathbf{p}^{j+1}$. Consequently computing the forward wavelet transform requires the solution of sparse linear systems. To solve these systems we use a bi-conjugate gradient solver [11] with diagonal preconditioning. We found the condition number for up to a 7 level transform to be no worse than 30 depending on the model.

Of course solving a linear system makes the forward transform slower than the inverse transform. This is acceptable as encoding is typically done once off-line while decoding happens frequently and in real time. For the Venus model the Loop forward transform, for example, takes 30s on a 550Mhz Pentium II Xeon while the inverse transform takes 2.5s. In case symmetry is important one can use a lifted Butterfly wavelet for which both forward and inverse transforms take about 2.5s.

The decorrelating power of the wavelet transform is illustrated in Figure 5. On the left is the histogram of the magnitude of Venus vertex positions. On the right is a histogram of the magnitude of the wavelet coefficients. Clearly a large amount of correlation was removed and the first order entropy has decreased considerably.

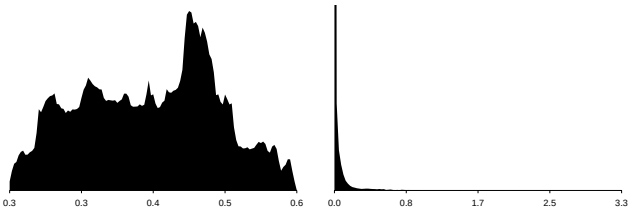


Figure 5: *Left: histogram of vertex position magnitudes for Venus. Right: histogram of the wavelet coefficient magnitudes, showing the decorrelation power of the wavelet transform.*

3.3 Vector Valued Wavelet Coefficients

Since our wavelet coefficients are vector valued, it is not immediately clear how they should be quantized. There is a fair amount of correlation between the x, y, and z wavelet components. We found that representing the wavelet coefficients in a local frame [39] induced by the surface tangent plane makes the components much more independent. In particular, we find that the variance of normal wavelet components is on average twice as large as the variance of the tangential components. Recalling the earlier geometry versus parameter distinction this is exactly what we want. In a smooth semi-regular mesh, the geometry information (normal component)

is much larger than the parameter information (tangential component). Figure 6 illustrates this by showing the histograms of the polar angles θ (the angle from the z of normal axis) of the wavelet coefficients in global and local coordinate frames. The distribution becomes very non-uniform in the local frame with peaks around 0 and π indicating that most of the wavelet vectors lie in the normal direction. The angle along the equator is fairly uniformly distributed both in the global and local frame, hence the choice of basis vectors in the tangent plane is not important. Recall that parameter,

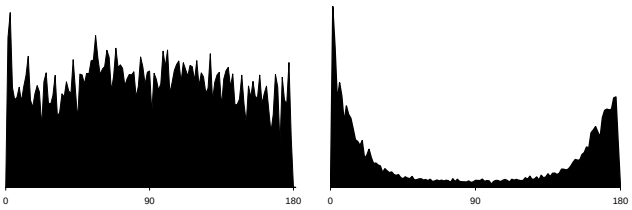


Figure 6: *Histograms of wavelet coefficient polar θ angles for the Venus head model in global (left) and local (right) frames. Coefficients lie mostly in the normal direction.*

i.e., tangential, information does not contribute to the error metric. Unfortunately, we cannot just ignore tangential wavelet components since this argument only holds in the infinitesimal limit. Especially at coarser levels, tangential wavelet coefficients can still contain some geometric information. However, we did find that the error metric is much less sensitive to quantization error of tangential versus normal wavelet components. Thus, we can further improve the error curves by more coarsely quantizing the tangential component.

A basic operation in a scalar zerotree coder is the *coefficient significance test*, i.e., checking its magnitude against a threshold. If it is below, the coefficient is added to a zerotree, else the location and sign of the coefficient need to be transmitted. For the vector case this becomes more difficult and we examined three quantization options. (1) Spherical cells are natural as we can use the magnitude for the significance test. We deal with the quantized angular components as "generalized" signs. (2) For cubical cells we divide the cube into 64 subcubes. Coefficients in the 8 internal cubes are insignificant and all the others are significant; their cell number again is an analog of the angular component. (3) We can deal with each vector component independently and encode it separately, reducing the vector case to three independent scalar passes.

We have compared all three cases and found that three scalar passes results in the best rate distortion curves for all models we considered. Experimentally, we found that quantization cells for the tangential component were best taken to be 4 times larger than those for the normal component.

3.4 Zerotree Coding

Given that we settled on scalar quantization, our coder consists of three independent zerotree coders. The bits from the three coders are interleaved to maintain progressivity.

A general principle of wavelet coefficient encoding is to send the highest order bits of the largest magnitude coefficients first. They will make the most significant contributions towards reducing error. Let $T_0 = \max\{|c_i|\}$ be the maximum magnitude of all coefficients, then in a first pass the coder should send the locations (index i) of *newly significant* coefficients, $|c_i| > T_0/2$. Doing so naïvely is expensive. However, if source and receiver agree on a canonical traversal order the source only has to send the result of the significance test $S(i) = (|c_i| > T)$ and, if true, the sign bit of c_i. If coefficients can be organized into canonical sets such that with high probability all coefficients in a given set are simultaneously below threshold, a few set-based significance tests can enumerate the locations of the relevant coefficients. The decay properties of wavelet coefficients make their hierarchical tree organization the natural set structure [32, 30, 5]. Coding consists of a number of passes with exponentially decreasing thresholds $T_{j+1} = T_j/2$. In each pass significance bits are sent for newly significant coefficients. Additionally, refinement bits are sent for those coefficients which became significant in an earlier pass. Since source and receiver already agreed on locations of the latter, no location bits have to be sent for them. The number of such bit plane passes depends on the final quantization level. The decoder can reconstruct the geometry associated with any prefix of the bitstream by running an inverse wavelet transform on the coefficient bits seen so far.

The main distinction of our setting from the image case is the construction of the zerotrees. For images, one associates the coefficients with a quadrilateral face and the trees follow immediately from the face quadtree. While this works also for dual, i.e., face based subdivision schemes, our triangular transform is primal, i.e, vertex based.

The main insight is that while scale coefficients are associated with vertices, wavelet coefficients have a one-to-one association with edges of the coarser mesh. Vertices do not have a tree structure, but the edges do. Each edge is the parent of four edges of the same orientation in the finer mesh as indicated in Figure 7. Hence, each edge of the base domain forms the root of a zerotree; it groups all the wavelet coefficients of a fixed wavelet subband from its two incident base domain triangles. The grouping is consistent for arbitrary semi-regular meshes, i.e., no coefficient is accounted for multiple times or left out.

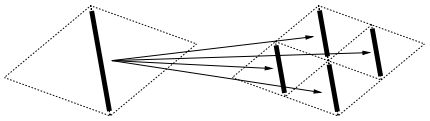


Figure 7: *A coarse edge (left) is parent to four finer edges of the same orientation (right).*

For brevity we do not give the complete coder/decoder algorithm here, but refer the interested reader to the pseudo code in [30], which is identical to our implementation with the above quadtree definition.

A final question concerns the transmission of the scale coefficients from the coarsest level. These are quantized uniformly. Experimentally, we found that it is best to send 4 bit planes initially with the base domain connectivity. Each remaining bitplane is sent as the zerotrees descend another bit plane.

The zerotree encoding (10 passes) of the Venus model takes 1s while decoding takes about 0.6s bringing the total decompression time to about 3.1s. Of course the low rate models can be decompressed faster.

3.5 Entropy Coding

The zerotree algorithm is very effective at exploiting parent-child coefficient correlations, minimizing the amount of encoded significance bits. However, the output of the zerotree coder can still be compressed further through arithmetic coding, which allows for a fractional number of bits per symbol.

The zerotree coder output contains three different types of information, significance bits, refinement bits and sign bits. Refinement and sign bits tend to be uniformly distributed; hence they are not entropy coded. Significance bits on the other hand can be further entropy coded. For early bitplanes most coefficients are insignificant resulting in mostly zero bits. For later bitplanes many coefficients become significant, resulting in mostly one bits. An arithmetic coder naturally takes advantage of this.

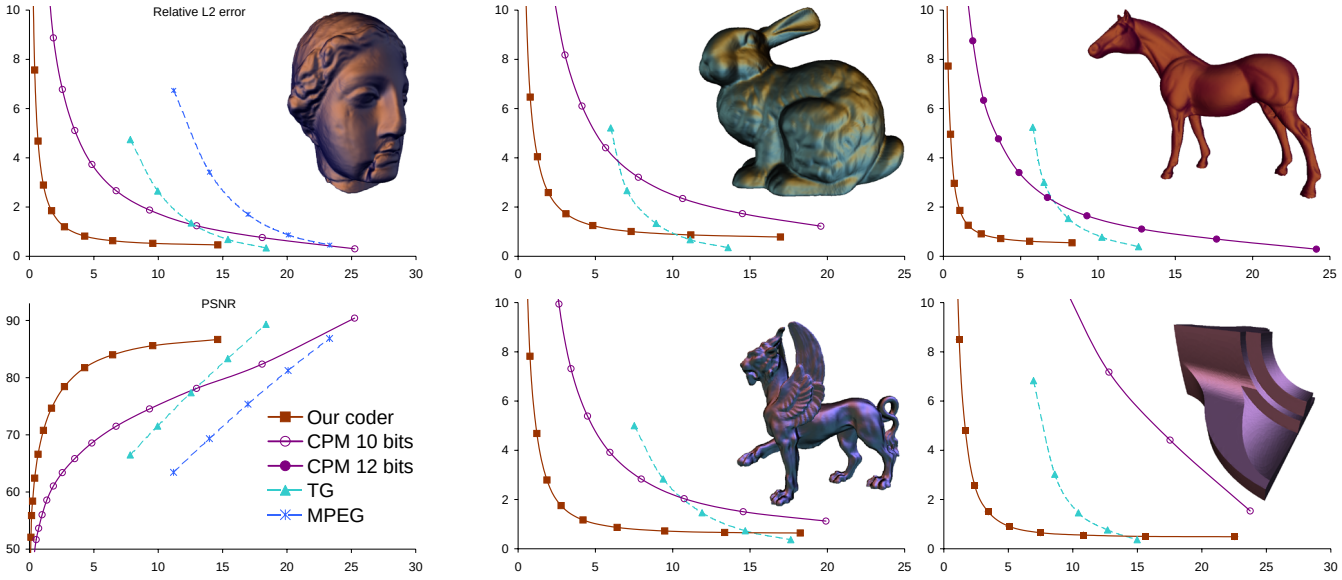


Figure 8: *Rate-distortion curves.*

We found that encoding of the significance bits in groups further improves performance of entropy coding [30]. Because children of any node always appear together during a zerotree pass we group their significance bits to form symbols of a 2^j alphabet ($j = 4, 3, 2, 1$). The actual number of bits of the alphabet is the number of children which were left insignificant at the previous pass. This grouping exploits correlations between magnitudes of spatially close wavelet coefficients.

4 Results

We compare our Loop based coder against known state of the art coders for different models. The coders we used are:

- **TG:** The Touma-Gotsman coder, which is a non progressive coder. It can be operated at different rates by changing the coordinate quantization between 8 and 12 bits.
- **CPM:** The compressed progressive mesh coder of Pajarola and Rossignac [26]. It can start with various quantization sizes. We found 10 or 12 to work best (and always show the best one).
- **MPEG:** The non-progressive coder from the MPEG4 standard which is based on topological surgery [35].

Figure 8 (left) shows the different curves for the Venus model for bitrates up to 25b/v. The top left shows relative L^2 error in units of 10^{-4}. The bottom left shows the same numbers but in a PSNR scale where PSNR $= 20 \log_{10} \text{peak}/d$, peak is the bounding box diagonal and d is the L^2 error. One can see that our progressive coder is about 12dB or a factor 4 better than the progressive CPM coder. As expected the non-progressive coders are much worse at lower rates and slightly better at higher rates. Our curve converges to the remeshing error which is where it crosses the TG curve. Given that the remeshing error is comparable to the discretization error, any compression with smaller error is only resolving a particular triangulation more accurately, but not increasing the geometric fidelity.

Figure 8 (right) shows the rate distortion curves for several additional models. Our curves are again significantly better. Typically the TG coder crosses our curve below the discretization error. For the fandisk, which is a model with creases, we used a tagged Loop transform which preserves the creases. The fandisk does not have that many triangles which is why the TG coder shows better rate-distortion performance than the CPM coder.

Figure 9 shows renderings of the different compressed versions of the model. This demonstrates the visual benefits of using subdivision based reconstruction. Note that the feline dataset has non-trivial genus (tail section), while the bunny has a number of holes on the bottom. For purposes of comparison (in the case of the Venus head) we have also rendered a number of partial bitstream reconstructions produced with the CPM coder (Figure 10) at file sizes comparable to our reconstructions (Figure 1). One could argue that the results of a more traditional progressive mesh coder could be improved by a smoothing post-process. However, even at very low bit rates, bit-plane progressivity in our coder implies that we see high order bits of significant coefficients at fine levels of the hierarchy early on. The resulting reconstructions always carry more detail than a straightforward Loop smoothing of some triangle mesh would capture. Finally Table 2 gives numerical error values for our coder at a variety of bit rates for the different models.

b/v	1/4	1/2	1	2	4	8
venus	15	6.1	3.1	1.60	0.85	0.55
feline	32	13	5.8	2.5	1.25	0.75
horse	9.7	4.5	2.0	1.05	0.70	0.55
bunny	22	10.8	5.1	2.5	1.40	0.95
fandisk		52	11.9	3.5	1.00	0.60

Table 2: *Relative L^2 error in units of 10^{-4} of our coder at various bitrates.*

5 Conclusion and Future Work

In this paper we described a progressive compression algorithm based on semi-regular meshes, wavelet transforms, and zerotree coders. Our rate distortion curves are significantly better than the best known progressive and non-progressive coders. This was achieved by explicitly treating sample locations and mesh connectivity as degrees of freedom of the coder. The progressive reconstructions especially at very low bit rates can be of astonishingly high visual quality.

There are several directions for future work:

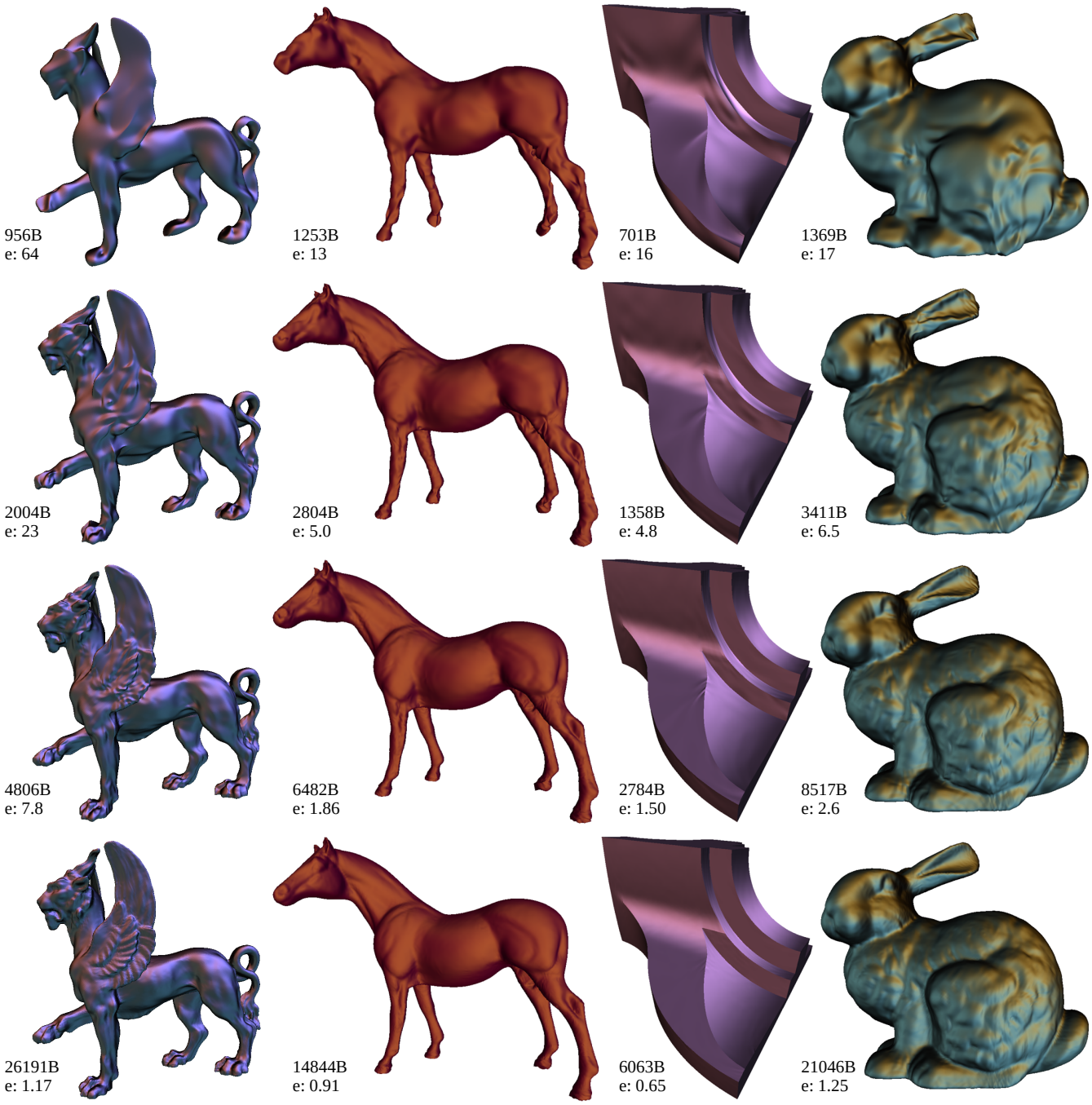


Figure 9: *File size in bytes and errors in units of* 10^{-4}.

- A mathematically sound theory for the construction of Loop wavelets around extraordinary vertices, including stability analysis.
- Construction of Loop wavelet transforms for adaptive semi-regular meshes. While all our reconstructions are performed adaptively, currently only lifted wavelets allow for adaptive analysis.
- Design of wavelet filters more suitable for geometry. Careful examination of reconstructed geometry reveals some ringing artifacts with our current wavelets.
- Even for our semi-regular meshes, there is still a fair amount of tangential information especially on the coarse levels. Recent work by Guskov et al. [15] shows that it is possible to construct *normal meshes*, i.e., meshes in which all wavelet coefficients lie exactly in the normal direction.
- The issues we discuss in this paper regarding geometry versus parameterization led to ideas such as coarsely quantizing the tangential components. These ideas can also be used to further improve irregular mesh coders.

Acknowledgments Andrei Khodakovsky was partially supported through an internship at Lucent Technologies. Other support came from NSF (ACI-9624957, ACI-9721349, DMS-9872890, DMS-9874082), Alias|Wavefront, a Packard Fellowship, and the SGI-Utah Visual Supercomputing Center. Special thanks to Cici Koenig,

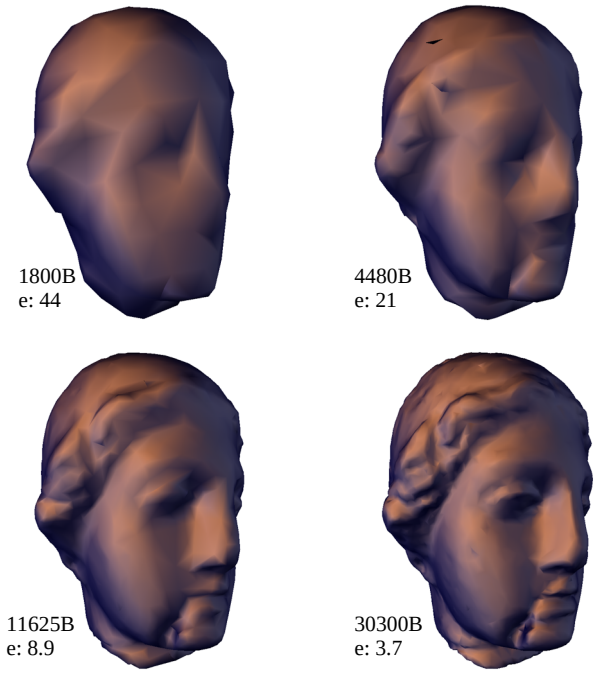


Figure 10: *Partial CPM reconstructions, error in units of* 10^{-4} *(compare to Fig. 1).*

Igor Guskov, Mathieu Desbrun, Aaron Lee, and Martin Vetterli. Datasets are courtesy Cyberware, the Stanford program in Computer Graphics and Hugues Hoppe. Our implementation uses an arithmetic coder of Geoff Davis and John Danskin. We are particularly grateful to Renato Pajarola, Craig Gotsman, and Gabriel Taubin for providing us with executables of their mesh compression algorithms.

References

[1] BAJAJ, C. L., PASCUCCI, V., AND ZHUANG, G. Progressive Compression and Transmission of Arbitrary Triangular Meshes. *IEEE Visualization '99* (1999), 307–316.

[2] CERTAIN, A., POPOVIC, J., DEROSE, T., DUCHAMP, T., SALESIN, D., AND STUETZLE, W. Interactive Multiresolution Surface Viewing. *Proceedings of SIGGRAPH 96* (1996), 91–98.

[3] CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum 17*, 2 (1998), 167–174.

[4] COHEN-OR, D., LEVIN, D., AND REMEZ, O. Progressive Compression of Arbitrary Triangular Meshes. *IEEE Visualization '99* (1999), 67–72.

[5] DAVIS, G., AND CHAWLA, S. Image Coding Using Optimized Significance Tree Quantization. In *Prodeedings Data Compression Conference*, 387–396, 1997.

[6] DAVIS, G., AND NOSRATINIA, A. Wavelet-based Image Coding: An Overview. *Applied Computational Control, Signals, and Circuits 1*, 1 (1998).

[7] DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow. *Proceedings of SIGGRAPH 99* (1999), 317–324.

[8] DEVORE, R. A., JAWERTH, B., AND LUCIER, B. J. Surface Compression. *Computer Aided Geometric Design 9* (1992), 219–239.

[9] DYN, N., LEVIN, D., AND GREGORY, J. A. A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control. *ACM Transactions on Graphics 9*, 2 (1990), 160–169.

[10] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. *Proceedings of SIGGRAPH 95* (1995), 173–182.

[11] GOLUB, G. H., AND LOAN, C. F. V. *Matrix Computations*, 2nd ed. The John Hopkins University Press, Baltimore, 1983.

[12] GROSS, M. H., STAADT, O. G., AND GATTI, R. Efficient Triangular Surface Approximations Using Wavelets and Quadtree Data Structures. *IEEE Transactions on Visualization and Computer Graphics 2*, 2 (1996).

[13] GUMHOLD, S., AND STRASSER, W. Real Time Compression of Triangle Mesh Connectivity. *Proceedings of SIGGRAPH 98* (1998), 133–140.

[14] GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. Multiresolution Signal Processing for Meshes. *Proceedings of SIGGRAPH 99* (1999), 325–334.

[15] GUSKOV, I., VIDIMCE, K., SWELDENS, W., AND SCHRÖDER, P. Normal Meshes. *Proceedings of SIGGRAPH 00* (2000).

[16] HOPPE, H. Efficient Implementation of Progressive Meshes. *Computers & Graphics 22*, 1 (1998), 27–36.

[17] KING, D., AND ROSSIGNAC, J. Optimal Bit Allocation in 3D Compression. Tech. Rep. GIT-GVU-99-07, Georgia Institute of Technology, 1999.

[18] KOBBELT, L., VORSATZ, J., LABSIK, U., AND SEIDEL, H.-P. A Shrink Wrapping Approach to Remeshing Polygonal Surfaces. *Computer Graphics Forum 18* (1999), 119 – 130.

[19] KOLAROV, K., AND LYNCH, W. Compression of Functions Defined on Surfaces of 3D Objects. In *Proc. of Data Compression Conference*, J. Storer and M. Cohn, Eds., 281–291, 1997.

[20] KRISHNAMURTHY, V., AND LEVOY, M. Fitting Smooth Surfaces to Dense Polygon Meshes. *Proceedings of SIGGRAPH 96* (1996), 313–324.

[21] LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. MAPS: Multiresolution Adaptive Parameterization of Surfaces. *Proceedings of SIGGRAPH 98* (1998), 95–104.

[22] LEVOY, M. The Digital Michelangelo Project. In *Proceedings of the 2nd International Conference on 3D Digital Imaging and Modeling*, October 1999.

[23] LI, J., AND KUO, C. Progressive Coding of 3-D Graphic Models. *Proceedings of the IEEE 86*, 6 (1998), 1052–1063.

[24] LOOP, C. Smooth Subdivision Surfaces Based on Triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.

[25] LOUNSBERY, M., DEROSE, T. D., AND WARREN, J. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *ACM Transactions on Graphics 16*, 1 (1997), 34–73. Originally available as TR-93-10-05, October, 1993, Department of Computer Science and Engineering, University of Washington.

[26] PAJAROLA, R., AND ROSSIGNAC, J. Compressed Progressive Meshes. Tech. Rep. GIT-GVU-99-05, Georgia Institute of Technology, 1999.

[27] RIEMENSCHNEIDER, S. D., AND SHEN, Z. Wavelets and Pre-Wavelets in Low Dimensions. *J. Approx. Th. 71*, 1 (1992), 18–38.

[28] ROSSIGNAC, J. Edgebreaker: Connectivity Compression for Triangle Meshes. *IEEE Transactions on Visualization and Computer Graphics 5*, 1 (1999), 47–61.

[29] ROSSIGNAC, J., AND SZYMCZAK, A. Wrap&Zip: Linear Decoding of Planar Triangle Graphs. Tech. Rep. GIT-GVU-99-08, Georgia Institute of Technology, 1999.

[30] SAID, A., AND PEARLMAN, W. A New, Fast, and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees. *IEEE Transaction on Circuits and Systems for Video Technology 6*, 3 (1996), 243–250.

[31] SCHRÖDER, P., AND SWELDENS, W. Spherical Wavelets: Efficiently Representing Functions on the Sphere. *Proceedings of SIGGRAPH 95* (1995), 161–172.

[32] SHAPIRO, J. Embedded Image-Coding using Zerotrees of Wavelet Coefficients. *IEEE Transactions on Signal Processing 41*, 12 (1993), 3445–3462.

[33] STAADT, O. G., GROSS, M. H., AND WEBER, R. Multiresolution Compression And Reconstruction. *IEEE Visualization '97* (1997), 337–346.

[34] TAUBIN, G., GUEZIEC, A., HORN, W., AND LAZARUS, F. Progressive Forest Split Compression. *Proceedings of SIGGRAPH 98* (1998), 123–132.

[35] TAUBIN, G., AND ROSSIGNAC, J. Geometric Compression Through Topological Surgery. *ACM Transactions on Graphics 17*, 2 (1998), 84–115.

[36] TAUBIN, G., AND ROSSIGNAC, J., Eds. *3D Geometry Compression*. No. 21 in Course Notes. ACM Siggraph, 1999.

[37] TOUMA, C., AND GOTSMAN, C. Triangle Mesh Compression. *Graphics Interface '98* (1998), 26–34.

[38] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interpolating Subdivision for Meshes with Arbitrary Topology. *Proceedings of SIGGRAPH 96* (1996), 189–192.

[39] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interactive Multiresolution Mesh Editing. *Proceedings of SIGGRAPH 97* (1997), 259–268.

Normal Mesh Compression

Andrei Khodakovsky
Caltech

Igor Guskov
Caltech

549B (e:54, p:45dB) 1225B (e:20, p:54dB) 3037B (e:8.1, p:62dB) 18111B (e:1.77, p:75dB) original

Figure 1: *Partial reconstructions from a progressive encoding of the dinosaur model. File sizes are given in bytes, errors in multiples of* 10^{-4} *and PSNR in dB. The right most model is the original which has 14K vertices (model courtesy of Cyberware).*

Abstract

Normal meshes were recently introduced as a new way to represent geometry. A normal mesh is a multiresolution representation which has the property that all details lie in a known normal direction and hence the mesh depends only on a *single scalar per vertex*. Such meshes are ideally suited for progressive compression. We demonstrate such a compression algorithm for normal meshes representing complex, arbitrary topology surfaces as they appear in 3D scanning and scientific visualization. The resulting coder is shown to exhibit gains of an additional 2-5dB over the previous state of the art.

1 Introduction

The unrelenting growth of computing power of personal computers and recent progress in shape acquisition technology facilitate the wide use of highly detailed meshes in industry and entertainment. Similarly, scientific visualization applications tend to produce ever finer meshes, such as iso-surfaces. In their raw, irregular form, acquired meshes are complex and often unmanageable due to their sheer size and irregularity. It is therefore important to find more efficient and compact representations. Algorithms for efficient encoding of such meshes have been described in both single rate [29, 9, 27, 22, 21] and progressive settings [19, 20, 16, 12, 3, 26, 1]. For an overview of 3D geometry compression see [28].

One should recognize the fact that compression is always a trade-off between size and accuracy. That is especially true for meshes that come from shape acquisition or iso-surface extraction, which always carry sampling error and acquisition noise. Hence the compression can be lossy as long as the approximation error is comparable with the sampling error. Recently, a number of efficient "remeshing" techniques have appeared that replace the original mesh with a mesh consisting of a number of "regular" pieces, such as B-spline [8], NURBS [14], or subdivision connectivity [15, 7] patches. Naturally, one should also expect that the remeshed model should behave much better with regards to compression algorithms. This expectation was confirmed in [13], where the MAPS algorithm [15] was included as part of a progressive geometry coder. In particular, the paper makes it clear that any mesh representation can be considered as having three components: geometry, connectivity, and parameterization; moreover, the last two components are not relevant for the representation of the geometry. For semi-regular mesh hierarchies, one can make the (reasonable) assumption that the normal component of the detail coefficients stores the geometric information, whereas the tangential components carry the parametric information (see also [10, 5]).

It is clear that many existing remeshing algorithms remove almost all of the connectivity information from the mesh, and also reduce the parametric information. One may wonder: *is it possible to get rid of the parametric information entirely?* The answer is yes as demonstrated in [11]. In particular, fixing the transform algorithm to use unlifted Butterfly wavelets it is possible to build a semi-regular mesh whose details lie in the local normal direction. Consequently, the geometry of "normal" meshes is fully represented by one scalar per vertex, instead of the usual three. Therefore it is natural to use normal meshes for compression as we do in this paper.

Contribution The goal of this paper is to demonstrate the additional coding gains possible when employing normal semiregular meshes rather than standard semiregular remeshes (such as those produced by MAPS [15]).

2 Compression Algorithms

The progressive geometry coding described in [13] requires three necessary components: a remeshing algorithm, a wavelet transform, and a zerotree coder. In [13] the MAPS remesher [15] was

used, followed by the Loop or Butterfly wavelet transform. In this paper, we are using the normal remesher of [11] which was specifically designed to produce detail coefficients with no tangential components when an *unlifted Butterfly* wavelet transform is applied to the produced semi-regular mesh. We use the same zerotree coder as in [13]. The following sections will briefly overview the algorithms used for compression. For a more detailed exposition, the reader is referred to [13] and [11].

2.1 Normal Meshes

The normal remesher starts with an arbitrary irregular mesh, and proceeds to build a semi-regular mesh hierarchy approximating the original model. The algorithm is described in [11] and consists of two stages. Using mesh simplification, a base mesh is chosen that is topologically equivalent to the original mesh. The connectivity of this base mesh will eventually become the connectivity of the coarsest level in the semi-regular hierarchy. Also at this stage a net of surface curves is initialized that splits the irregular mesh into a number of non-intersecting regions (these regions are in a one-to-one correspondence with the coarsest level faces of the semi-regular mesh we are constructing). Next, the net of surface curves is propagated to the finest level of the original irregular mesh, and a relaxation of global vertex positions within the surface is performed to even out their distribution and improve aspect ratios of the base mesh triangles.

In the second stage of the algorithm, a "piercing procedure" is applied recursively to obtain positions of finer level points of the semi-regular mesh. Thus, the semi-regular mesh is refined and, to maintain the status quo, the corresponding regions of the mesh are split into smaller subregions. A global adaptive parameterization is maintained on the original mesh in order to keep the piercing process under control and to enable fast construction of surface curves. The described two-stage process produces a semi-regular mesh that has mostly normal detail coefficients except for a small number of locations where the piercing did not find any "valid" intersection point with the original surface. The percentage of non-normal coefficients varies depending on the geometric properties of a given mesh and the corresponding coarse level points chosen in the first stage of the algorithm. Typically, the number of non-normal coefficients is below 10% for adaptive meshes that have the same number of vertices as the original irregular mesh. For a detailed description of the algorithm refer to [11].

2.2 Wavelet Transform

The wavelet transform replaces a fine semi-regular mesh with some coarsest level base mesh and a sequence of wavelet coefficients expressing the difference between successive levels of the mesh hierarchy. In [13] a novel Loop [17] wavelet transform was described. It has the advantage that the inverse transform uses Loop subdivision. Several other methods for building wavelet transforms on semi-regular meshes exist [18, 24]. In this work we use the unlifted version of Butterfly wavelets [6, 30] because *exactly* the same transform is used to produce normal meshes. A detailed description of the construction of Butterfly wavelets can be found in [24].

Note that the Butterfly wavelet transform uses finite filters for both analysis and reconstruction, and is therefore faster than the forward Loop transform which requires solving a sparse linear system. The Loop reconstruction filter has support of the same size as the Butterfly filter. On the other hand, we found that the Loop wavelet transform typically yields better visual appearance than the Butterfly transform, though comparable error measures.

Typically, the x, y, and z wavelet components are correlated [13]. We exploit this correlation by expressing wavelet coefficients in local frames induced by the coarser level. This is especially true for normal meshes. Almost all wavelet coefficients computed for normal meshes have only a normal component. Figure 2 shows histograms of the latitude angles θ (the angle from the normal axis) of the Butterfly and Loop wavelet coefficients in a local coordinate frame. Since the normal mesh is built using butterfly subdivision almost all Butterfly coefficients have only normal components.

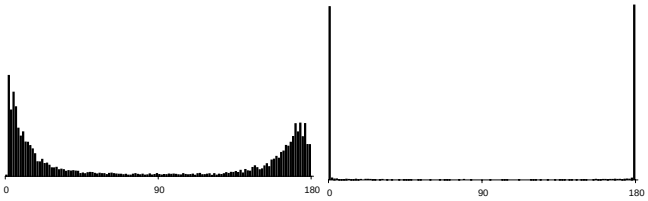


Figure 2: *Histograms of the Loop (left) and Butterfly (right) wavelet coefficients latitude θ angle for the Venus head model in a local frame.*

2.3 Zerotree Coding

We encode components of wavelet coefficients separately, that is, our coder essentially consists of three independent zerotree coders. The bits from the three coders are interleaved to maintain progressivity.

A general principle of wavelet coefficient encoding is to send the highest order bits of the largest magnitude coefficients first. They will make the most significant contributions towards reducing error. The zerotree algorithm [25, 23, 4] groups all coefficients in hierarchical zerotree sets such that with high probability all coefficients in a given set are simultaneously below threshold.

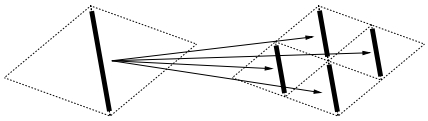


Figure 3: *A coarse edge (left) is parent to four finer edges of the same orientation (right).*

The main distinction of our setting from the image case is the construction of the zerotrees. For images, one associates the coefficients with a quadrilateral face and the trees follow immediately from the face quadtree. For semi-regular mesh hierarchies, the main insight is that while scale coefficients are associated with vertices, wavelet coefficients have a one to one association with edges of the coarser mesh. Vertices do not have a tree structure, but edges do. Each edge is the parent of four edges of the same orientation in the finer mesh as indicated in Figure 3. Hence, each edge of the base domain forms the root of a zerotree; it groups all the wavelet coefficients of a fixed wavelet subband from its two incident base domain triangles.

The scale coefficients from the coarsest level are quantized uniformly and also progressively encoded. Each bitplane is sent as the zerotree descends another bit plane. Finally, the output of the zerotree algorithm is encoded using an arithmetic coder.

3 Adaptive Reconstruction

There is a trade-off between remeshing error and the size of the resulting semi-regular mesh. With adaptive remeshing this trade-off is local. We refine the mesh where the error is maximal and leave it coarse where the remeshing error is small (Figure 5). Note, that the coarse remesh and its refinement give the same compression

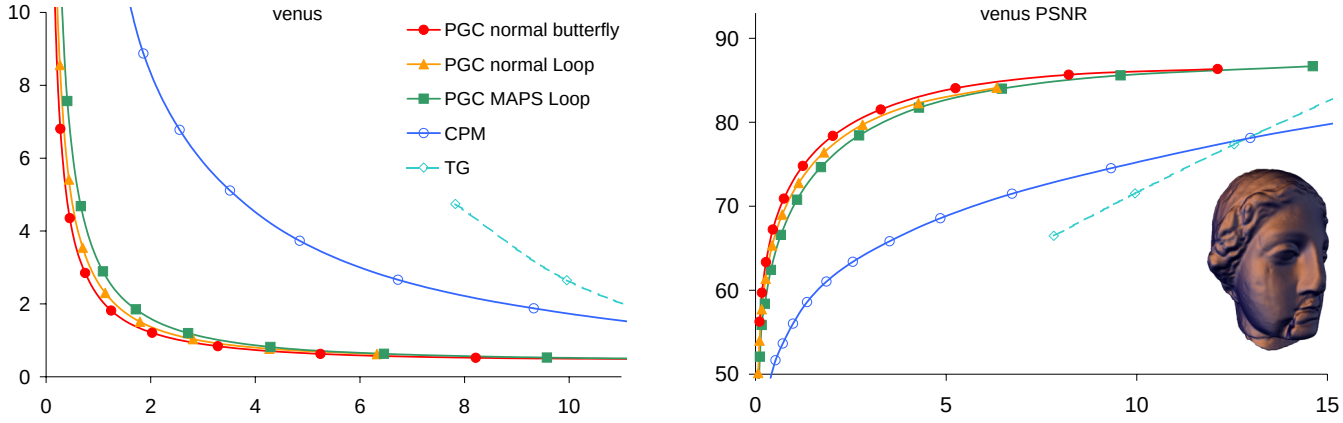


Figure 4: *Rate distortion for our coder with Butterfly and Loop wavelets using MAPS and normal meshes, TG, and CPM coders for the Venus model. On the left relative L^2 error in multiples of 10^{-4} as a function of bits/vertex. On the right the PSNR in dB.*

performance at low bit-rates, since refinement usually introduces small details at finer levels. These details are ignored by the zerotree coder at low bit-rates.

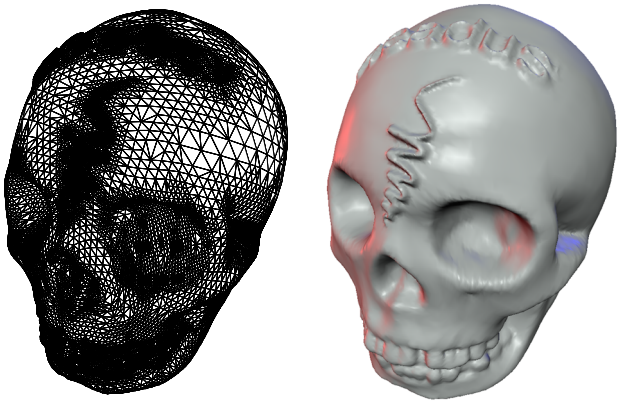


Figure 5: *Adaptive normal mesh for the skull (19138 vertices) with relative L^2 error of 0.02% and relative L^∞ error of 0.09%. The base mesh is a tetrahedron (4 vertices) while the original mesh has 20002 vertices (model courtesy of Headus).*

The compression algorithm is also adaptive. All regions which are not subdivided to the finest level define zero wavelet coefficients. These coefficients "virtually" exist as an extension of non-uniform zerotrees. The zerotree coder is naturally adaptive: There is no difference between non-uniform and uniform zerotrees with zeros attached to extra nodes of the latter. During the reconstruction step we subdivide faces only when we decode a wavelet coefficients which belongs to those faces. After we decode all coefficients we subdivide all faces until we meet an adaptive flatness threshold [31]. This step is important since we want to produce smooth surfaces even at low bit-rates. The flatness threshold is controlled by the user and can be chosen depending on the reconstruction bit-rate.

4 Results and Discussion

We measured the performance of our coder [13] with normal meshes using both Butterfly and Loop wavelet transforms. For comparison we use the CPM coder of Pajarola and Rossignac [19] and the single-rate coder of Touma and Gotsman [29]. We plotted rate-distortion curves for Touma-Gotsman coder by changing the coordinate quantization between 8 and 12 bits. Also, we compare our results with the performance of the coder of [13] for the Venus, horse and rabbit models.

Error was measured using the publicly available METRO tool [2]. All graphs show error between the reconstruction and the irregular original model as a function of the number of bits per vertex of the original mesh. All errors are given as PSNR (PSNR$= 20 \log_{10}$ BBoxDiag$/L^2$-error).

	Venus	rabbit	horse	dinosaur	skull	molecule
V_{or}	50K	67K	48K	14K	20K	10K
V_b	42	71	112	128	4	53
$E_r, 10^{-4}$	0.47	0.47	0.51	1.7	2.3	6.3
r_n	1580	759	754	2973	1861	794

Table 1: *Number of vertices in original models (V_{or}), base domain of remeshes (V_b), relative L^2 error (E_r) in units of 10^{-4}, and the number of non-normal coefficients (r_n).*

We found that the coding with normal meshes has better performance compared to MAPS meshes. Even using a Loop wavelet transform we observe an improvement. For example, for the horse model Loop (Figure 6) wavelets on the normal mesh allow 1.5 times (3.5dB) smaller distortion than on the MAPS mesh. Note, that an improvement in the high bit-rate limit is due to the smaller remeshing error. More important is that we have better distortion for all bit-rates which happens because of the normality of the mesh. The additional improvement of about a factor of 1.2 (1.5dB) comes from using Butterfly wavelets summing up to a total of 5dB improvement compared to [13]. For the rabbit and Venus models MAPS meshes are closer to normal so the coders using MAPS and normal meshes have closer performance. However we still observe an improvement of at least 2dB.

The fact that encoding with Loop wavelets benefits from Butterfly normal meshes is remarkable. Experimentally we found that if we treat wavelet components completely separately, Loop normals compress better than Butterfly normals. Given this evidence we should expect further improvement with Loop based normal meshes.

Figure 7 shows rate-distortion curves for the dinosaur, skull, and molecule models. These have coarser original meshes, therefore we allow a larger remeshing error (see the discussion on the natural choice of remeshing error in [13]). Note, that the remesh of the skull model has less vertices than the original, and has a reasonably small remeshing error. The base domain for the skull is a tetrahedron. For all the meshes mentioned in this paper, the remeshing step of our algorithm takes less than 10 minutes to complete. The forward Loop transform of the coder requires the solution of a sparse linear system, which takes about 30 seconds. Loop reconstruction and both Butterfly forward and backward transforms take 2-3 sec-

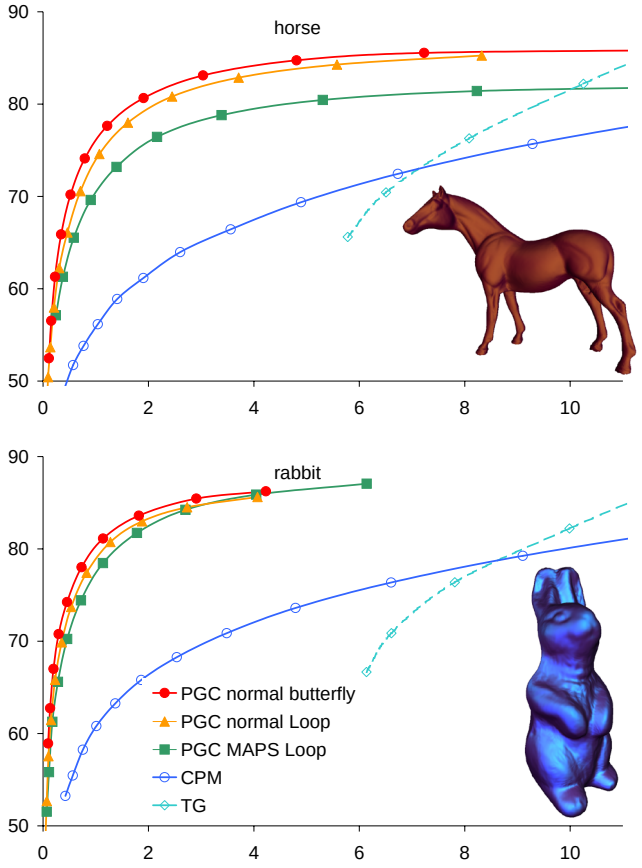


Figure 6: *Rate-distortion for rabbit and horse models showing PSNR as a function of bits/vertex.*

onds. Zerotree encoding and decoding take about 1 second.

5 Conclusions

In this paper we show that normal meshes improve performance of progressive geometry compression. We observe improvement of 2-5 dB depending on the model. Also we described adaptive compression which allows finer control on the number of vertices in the reconstructed model.

One of the directions for the future work is the design of normal meshes using Loop wavelets.

Acknowledgments This work was supported in part by NSF (ACI-9624957, ACI-9721349, DMS-9874082). The paper was written with great encouragement and invaluable help from Wim Sweldens and Peter Schröder. Kiril Vidimče contributed to the current implementation of normal remesher. Datasets are courtesy Cyberware, Headus, The Scripps Research Institute, and University of Washington.

References

[1] Bajaj, C. L., Pascucci, V., and Zhuang, G. Progressive compression and transmission of arbitrary triangular meshes. *IEEE Visualization '99* (1999), 307–316.

[2] Cignoni, P., Rocchini, C., and Scopigno, R. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2 (1998), 167–174.

[3] Cohen-Or, D., Levin, D., and Remez, O. Progressive compression of arbitrary triangular meshes. *IEEE Visualization '99* (1999), 67–72.

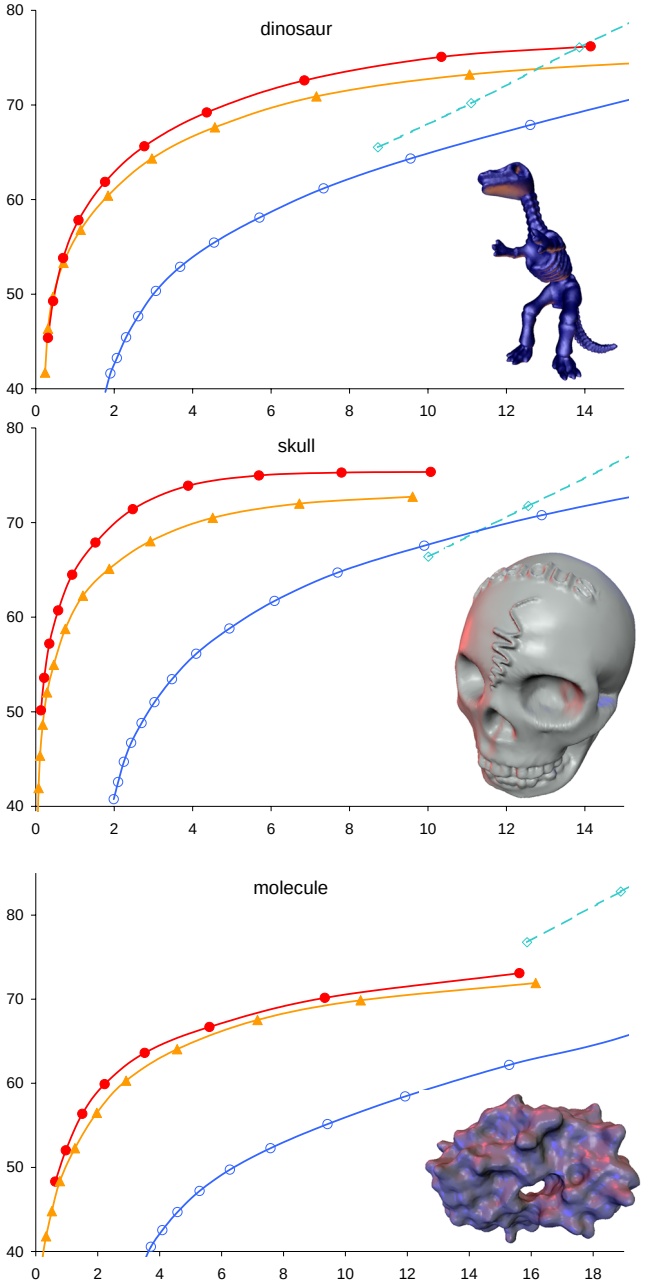


Figure 7: *Rate-distortion for our coder with Butterfly and Loop wavelets using normal meshes, TG and CPM coders for dinosaur, skull, and molecule models in PSNR as a function of bits/vertex.*

[4] Davis, G., and Chawla, S. Image coding using optimized significance tree quantization. In *Prodeedings Data Compression Conference* (1997), IEEE, pp. 387–396.

[5] Desbrun, M., Meyer, M., Schröder, P., and Barr, A. H. Implicit fairing of irregular meshes using diffusion and curvature flow. *Proceedings of SIGGRAPH 99* (1999), 317–324.

[6] Dyn, N., Levin, D., and Gregory, J. A. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics 9*, 2 (1990), 160–169.

[7] Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., and Stuetzle, W. Multiresolution analysis of arbitrary meshes. *Proceedings of SIGGRAPH 95* (1995), 173–182.

[8] Eck, M., and Hoppe, H. Automatic reconstruction of b-spline surfaces of arbitrary topological type. *Proceedings of SIGGRAPH 96* (1996), 325–334.

[9] GUMHOLD, S., AND STRASSER, W. Real time compression of triangle mesh connectivity. *Proceedings of SIGGRAPH 98* (1998), 133–140.

[10] GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. Multiresolution signal processing for meshes. *Proceedings of SIGGRAPH 99* (1999), 325–334.

[11] GUSKOV, I., VIDIMČE, K., SWELDENS, W., AND SCHRÖDER, P. Normal meshes. *to appear in Proceedings of SIGGRAPH 2000* (2000).

[12] HOPPE, H. Efficient implementation of progressive meshes. *Computers & Graphics 22*, 1 (1998), 27–36.

[13] KHODAKOVSKY, A., SCHRÖDER, P., AND SWELDENS, W. Progressive geometry compression. *to appear in Proceedings of SIGGRAPH 2000* (2000).

[14] KRISHNAMURTHY, V., AND LEVOY, M. Fitting smooth surfaces to dense polygon meshes. *Proceedings of SIGGRAPH 96* (1996), 313–324.

[15] LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. Maps: Multiresolution adaptive parameterization of surfaces. *Proceedings of SIGGRAPH 98* (1998), 95–104.

[16] LI, J., AND KUO, C. Progressive coding of 3-d graphic models. *Proceedings of the IEEE 86*, 6 (1998), 1052–1063.

[17] LOOP, C. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.

[18] LOUNSBERY, M., DEROSE, T. D., AND WARREN, J. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics 16*, 1 (1997), 34–73. Originally available as TR-93-10-05, October, 1993, Department of Computer Science and Engineering, University of Washington.

[19] PAJAROLA, R., AND ROSSIGNAC, J. Compressed progressive meshes. Tech. Rep. GIT-GVU-99-05, Georgia Institute of Technology, 1999.

[20] PAJAROLA, R., AND ROSSIGNAC, J. SQUEEZE: Fast and progressive decompression of triangle meshes. In *Proc. CGI* (2000).

[21] ROSSIGNAC, J. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics 5*, 1 (1999), 47–61.

[22] ROSSIGNAC, J., AND SZYMCZAK, A. Wrap&zip: Linear decoding of planar triangle graphs. Tech. Rep. GIT-GVU-99-08, Georgia Institute of Technology, 1999.

[23] SAID, A., AND PEARLMAN, W. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transaction on Circuits and Systems for Video Technology 6*, 3 (1996), 243–250.

[24] SCHRÖDER, P., AND SWELDENS, W. Spherical wavelets: Efficiently representing functions on the sphere. *Proceedings of SIGGRAPH 95* (1995), 161–172.

[25] SHAPIRO, J. Embedded image-coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing 41*, 12 (1993), 3445–3462.

[26] TAUBIN, G., GUEZIEC, A., HORN, W., AND LAZARUS, F. Progressive forest split compression. *Proceedings of SIGGRAPH 98* (1998), 123–132.

[27] TAUBIN, G., AND ROSSIGNAC, J. Geometric compression through topological surgery. *ACM Transactions on Graphics 17*, 2 (1998), 84–115.

[28] TAUBIN, G., AND ROSSIGNAC, J., Eds. *3D Geometry Compression*. No. 21 in Course Notes. ACM Siggraph, 1999.

[29] TOUMA, C., AND GOTSMAN, C. Triangle mesh compression. *Graphics Interface '98* (1998), 26–34.

[30] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interpolating subdivision for meshes with arbitrary topology. *Proceedings of SIGGRAPH 96* (1996), 189–192.

[31] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interactive multiresolution mesh editing. *Proceedings of SIGGRAPH 97* (1997), 259–268.